IMPROVING ROBUSTNESS OF DNS GRAPH CLUSTERING AGAINST NOISE

A Dissertation
Presented to
The Academic Faculty

By

Yizheng Chen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology

December 2017

IMPROVING ROBUSTNESS OF DNS GRAPH CLUSTERING AGAINST NOISE

Approved by:

Dr. Manos Antonakakis, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Wenke Lee
School of Computer Science
Georgia Institute of Technology

Dr. Roberto Perdisci
Department of Computer Science
University of Georgia

Dr. Mustaque Ahamad
School of Computer Science
Georgia Institute of Technology

Dr. Raheem Beyah
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: October 13, 2017

To my parents,

for all the support and love.

ACKNOWLEDGEMENTS

First of all, I am deeply grateful for my advisors Dr. Manos Antonakakis and Dr. Wenke Lee. Dr. Manos Antonakakis showed me the best quality a great researcher could have: never get disappointed. He believed in me when I did not believe in myself. The guidance and support from Dr. Wenke Lee taught me critical thinking and broader views of making a contribution to the security research community.

I would like to thank my thesis committee members Dr. Roberto Perdisci, Dr. Mustaque Ahamad, and Dr. Raheem Beyah for taking time to serve on my thesis committee. Their insightful feedback has helped me improve this thesis tremendously.

I have been fortunate to work with some brilliant people during my Ph.D. study. Dr. Yacin Nadji is always available for brainstorming ideas, even by sacrificing his own time. I cannot thank him enough for his help. Panagiotis Kintis has helped with many important experiments of research projects, and he is the most reliable teammate one can find. This dissertation would not be possible without the help from Dr. Nikolaos Vasiloglou, Dr. Fabian Monrose, and David Dagon. In addition, I would like to thank Astrolavos colleagues Chaz Lever, Athanasios Kountouras, Rosa Romero-Gómez, and Omar Alrawi.

I am grateful for the inspiring collaborators and colleagues: Dr. Tielei Wang, Dr. Simon P. Chung, Dr. Yeongjin Jang, and Billy Lau; my internship mentors, Dr. Vinod Yegneswaran and Dr. Nektarios Leontiadis for helping me become a better researcher.

My friends have helped me survive the most difficult times in the Ph.D. program. Special thanks to Qiaoyi Xu, Dr. Wei Meng, Dr. Byoungyoung Lee; my cooking night crew: Dr. Adrian Lam, Tiffany Tse, Dr. Samantha Lo, Dr. Brian Gestner, Dr. Samson Lai, Dr. Elaine Tang, Dr. Jeff Wong, Wenny Liu, Dr. Leung Kin Chiu, Wesley and others.

Last but not least, I must thank my parents for their unconditional love and support to pursue my dream.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Clustering is often the first step performed to assist us in finding structure within unlabeled datasets. Given a small set of labels, clustering also enables us to propagate these labels by discovering groups of objects that are similar to each other. The ever-growing amount of data being collected over a long period of time brings us many challenging opportunities to conduct clustering. Analyzing such long-term datasets allows us to solve evolving security problems such as: botnet forensic analysis; early warning of new threats; and the evolution of security phenomena. However, the analysis also faces the challenge presented by noise in the data.

This thesis improves the robustness of clustering against noise by focusing on DNS graphs. Noise is either inherent in the dataset, or can be injected by adversaries. The first goal of the thesis is to remediate the effect of the noise inherent in the data. To that end, we perform measurement studies from two different vantage points in the online advertising ecosystem. As a multi-billion dollar industry, the online ad ecosystem naturally attracts ad abuse from miscreants. We propose a new clustering technique to automatically analyze the costs of impression fraud to advertisers generated by the botnet TDSS/TDL4 over four years. In addition, our measurement results show statistically significant differences between blacklisted publishers compared to those that were never blacklisted, from the vantage point of a Demand Side Platform provider.

The second goal of the thesis is to increase the robustness of clustering against adversarial noise. Little work has been done in adversarial clustering in order to understand the weaknesses of clustering systems. We propose two novel attacks, one that injects noise to existing clusters, and one that moves data points to noisy clusters. After analyzing the effectiveness and the cost of attacks, we present defense techniques that improve the robustness of clustering in adversarial settings.

CHAPTER 1
INTRODUCTION

Datasets are being generated at an unprecedented rate, which brings challenges to analyze them. Nowadays, large corporations collect and store Petabytes of data [1, 2]. With smartphones and the internet-of-things devices generating a wide variety of new data, the increase in dataset size is only going to become more significant. Finding meaningful information from massive amounts of data requires considerably more effort, especially when the datasets are entirely or largely unlabeled.

Clustering is often the first step taken in an effort to find structure within unlabeled datasets. Additionally, when there is a small set of available seed labels, clustering may also be performed to propagate the labels. Repeating this process over time can help us understand the evolution of threats. For example, clustering long-term datasets enables botnet forensic analysis, provides early warning of new threats, and tracks the evolution of security phenomena. Performing such an analytical process reliably requires solutions for the challenge presented by noise. This noise can be inherent in the data, or injected by adversaries. The ability to handle noise is thus essential to perform clustering at scale.

This thesis aims to improve the robustness of clustering against noise by analyzing DNS graphs. We mine two types of DNS bipartite graphs: 1) the domain names resolution graph that represents Internet hosting infrastructure, i.e., passive DNS datasets; and 2) the graph of hosts querying domain names.

The first goal of the thesis is to deal with the inherent noise in the dataset. For example, in the passive DNS graph, domains can point to noisy internet infrastructure if the owners decide to "park" them. In the graph of hosts querying domains, infected hosts carry out a variety of benign activities, which adds noise to malicious behavior. We propose a new clustering technique that is robust against inherent noise in the DNS graph.

We apply our novel technique to measure the impression fraud generated by the TDSS/TDL4 botnet. Impression fraud is a severe problem in the online advertising ecosystem, which is a multi-billion dollar industry. While most efforts have been focused on remediating click fraud, the online advertising industry has only just started to draft standards [3, 4] for detecting fraudulent impressions. For defenders, impression fraud is a significantly harder problem to solve compared to click fraud. Therefore, conducting impression fraud is a high-return and low-risk monetization method employed by attackers. Using our robust clustering technique, we are able to automatically measure the lower-bound loss of advertisers caused by TDSS/TDL4, over four years of the botnet's lifetime. In comparison, related work either relies on manual effort undertaken by law enforcement [5, 6], or only limits the study to small time periods such as two weeks [7].

Our TDSS/TDL4 study provides a vantage point of infected machines, which is outside of the ad ecosystem. We also study the ad abuse that can be observed from within the ad ecosystem, from the vantage point of Demand Side Platform (DSP) providers. Our analysis shows that traditional blacklists can be used to understand malicious publishers seen within the ad ecosystem. However, deploying blacklists is not sufficient due to very small overlap between blacklists and publishers. Since the few blacklisted publishers do not associate with noisy hosting infrastructure on the DSP graph, we use a simple graph connected component analysis to track malicious publishers. In addition, our measurement results show that the behavior of blacklisted publishers differs significantly from those that have never been blacklisted. Therefore, building a reputation system is possible within the ad ecosystem.

The second goal of the thesis is to make clustering systems more robust against adversarial noise. If a detection system is deployed, attackers will try to evade it. Many researchers have shown that classifiers can be evaded [8, 9, 10, 11, 12, 13, 14, 15, 13, 16, 17, 18]. On the other hand, limited attention has been paid to adversarial clustering [19, 20]. We face the challenge that the result of clustering depends on all the data points being

clustered. By contrast, related work can compute classification features directly from one data point, e.g., image, PDF, phishing page, network packet, or exploit. Therefore, we have to consider attackers with different knowledge levels in our threat model.

In adversarial settings, an attacker can either inject noise to existing clusters, or she can move meaningful data points to noisy clusters. We propose two novel attacks that generate the two types of adversarial noise. Furthermore, we analyze the cost of the attacks, and present methods to increase the cost for the attackers to evade clustering. Our adversarial clustering analysis can identify the weaknesses of a clustering system, which enables us to improve the system for the defender.

1.1 Thesis Contributions

This thesis makes the following technical contributions:

- **New Clustering Technique:** We develop a spectral expansion technique that is robust against inherent noise in the data. Our technique can reliably extend the set of botnet seed domains from a few days of ground truth to four years.

- **New Measurement Results in DSP:** Our measurement results show that malicious ad campaigns have statistically significant differences in traffic and lookup patterns from benign ones. These new findings suggest that reputation systems for advertisement publishers are possible.

- **New Adversarial Clustering Analysis:** We present two novel attacks against graph-based clustering, and two defense techniques that reduce the effectiveness of the attacks. The attacks can be used to improve the robustness of the clustering system against adversarial noise.

1.1.1 Financial Lower Bounds of Online Advertising Abuse

From the edge of the online ad ecosystem, we develop the Ad-abuse Analysis System (A^2S), which is able to analyze one of the most complex, sophisticated, and long-lived botnets: TDSS/TDL4. The ad-abuse module of TDSS/TDL4 uses a server-side DGA algorithm to generate related C&C domains. The malware does not contain the DGA algorithm, but receives the domains in config files from the C&C server. The bots receive commands to conduct ad abuse, which causes advertisers to lose money.

The goal of A^2S is to estimate lower bounds of the advertisers' financial loss caused by the botnet using data-driven approaches. With this knowledge, network operators, such as large Internet Service Providers (ISPs), can design network policies to reduce both (1) the economic gains for adversaries that monetize ads and (2) the overall impact a botnet may have to the online ad ecosystem and the advertisers. We develop the spectral expansion module in A^2S to reliably extend the seed of ad-abuse C&C domains. After running the spectral expansion algorithm 2,590 times, we increased the set of ad-abuse C&C domains to almost four times the size of the seed set, with a very low (3 out of 838) false positive rate.

Using four years of network datasets from one of the largest ISPs in North America, we study: (1) the network infrastructure necessary to support the ad-abuse operation and (2) the financial model to estimate abuse the botnet inflicts on advertisers. Our major findings include:

- Online advertisers lost at least US$346 million to TDSS/TDL4. This amount is based solely on the actions of less than 15% of the botnet population. This translates to more than US$340 thousand per day on average, and the abuse was mostly accomplished by impression fraud. It is worth noting that daily abuse levels are three times of recent results reported for the ZeroAccess botnet [7] and as large as ten times of the short-lived DNSChanger [21] botnet.

- With respect to the infrastructure that supported this botnet operation, adversaries employed a level of network agility to achieve monetization similar to traditional botnet C&C communication. *At least* 228 IP addresses and 863 domain names were used to support the ad-abuse operation over four years. The domain names are available here [22].

1.1.2 Measuring Network Reputation in the Ad-Bidding Process

From within the online ad ecosystem, we investigate the potential of using public threat data to measure and detect adware and malicious affiliate traffic from the perspective of Demand Side Platforms (DSP). A DSP facilitates ad bidding between ad exchanges and advertisers. In summary, we found that:

- There are 13,324 (0.27%) known malicious domains generating bid request traffic through the ad exchanges in our datasets. On average, they generate 1.8% of the overall bid requests daily, much less than previously published values [23, 24]. However, we can use public blacklists to identify 68.28% of domains before they appeared in DSP traffic. This suggests traditional sources of maliciousness are valuable, but insufficient to fully understand ad-abuse from the perspective of the DSPs.

- On average, blacklisted publisher domains tend to use more ad exchanges (average: 1.85) and reach more clients (average: 5109.47) compared to non-blacklisted domains (average ad exchanges: 1.43, average hashed client IP addresses: 568.78). This suggests that reputation systems for ad publishers are possible.

- Contrary to the observation of blacklisted publisher domains, malware domains use a similar number of ad exchanges (average: 1.44), but are seen from more hashed client IP addresses (average: 2310.75), compared to publisher domains that are never queried by malware (average ad exchanges: 1.43, average hashed client IP addresses: 485.36).

Furthermore, we can use simple graph analysis and maliciousness heuristics to track malicious infrastructure observed by ad exchanges. Among the campaigns ranked the highest (top 0.1%), we found new cases including PUP, DGAs and malware sites.

1.1.3 Adversarial Analysis of Graph-based Detection System

We present the first practical attempt to attack graph-based modeling techniques in the context of network security. To that end, we devise two novel attacks (namely, targeted noise injection and small community attacks) against three commonly used graph clustering or embedding techniques, namely; i) Community Discovery, ii) Singular Value Decomposition (SVD), and iii) node2vec. Using three different classes of real world datasets (a US telecommunication dataset, a US university dataset and a threat feed) and after considering three classes of adversaries (adversaries with minimal, moderate and perfect knowledge) we mount these two new attacks against the graph modeling component of a state of the art network detection system: specifically, Pleiades [25]. We use the classifier model to test whether and how likely each cluster belongs to the real DGA malware family, both before and after the attack. Then, we present the overall distribution of such *predicted class probabilities* to evaluate the impact of the attacks.

The *targeted noise injection attack* injects vertices and edges to copy the graph structure of the original signal, which forces noise into the resulting clusters. In minimal knowledge, we create a DGA algorithm that can be classified as benign, effectively evading the classifier, to generate noisy domains for injection. In moderate knowledge, we use unpopular domains from a different network as noise. In perfect knowledge, we use unpopular domains from the same network traffic as noise. While more knowledgeable attackers typically fare better, we demonstrate that even minimal knowledge attackers are strong. Attackers with no knowledge beyond their infections can render the predicted class probabilities of 84% of the new clusters drops to zero. The attacks can be performed at a low cost to the adversary by not appearing to be anomalous. The majority of hosts had little change in "suspicious-

ness", whereas a small percentage of hosts increased their suspiciousness after the targeted noise injection attacks.

Our *small community attack* abuses the known property of small communities in graphs to subdivide and separate clusters into one or more unrelated clusters. Community discovery is resistant to the small community attack due to the high costs it would cause the attacker, however, spectral methods and node2vec are both vulnerable to the small community attack. We measure the cost of attacks by the decrease in the attacker graph density. Node2vec offers more adversarial resistance than SVD because the attack cost is higher.

We propose two defense techniques that could help Pleiades retain its detection capabilities — with the respect of the two proposed attacks. The first one is training the classifier with noise, which shows promise in remediating the noise injection attack to some extent. The second one is using the small community attack as an adversarial guideline to choose better hyperparamaters for graph embeddings, which can lower the attack success rate from 75% to 25%.

1.2 Dissertation Overview

In Chapter 2, we introduce different graph clustering methods that we will use in the thesis. We also describe the online advertising ecosystem and discuss the vantage points for Chapter 3 and Chapter 4.

In the first part of this dissertation (Chapter 3 and Chapter 4), we study impression fraud from both outside and inside the online advertising ecosystem. Specifically, we propose a new clustering technique to measure lower bound of advertiser's loss due to impression fraud generated by the botnet TDSS/TDL4, in Chapter 3. Our new technique generates very low false positives, and thus can improve the robustness of clustering against inherent noise in the data. In Chapter 4, we use clustering to track malicious publishers observed by a DSP, and also perform measurement study for the publisher reputation. Our results point to the direction of promising features for building a publisher reputation system.

In the second part of this dissertation (Chapter 5), we improve the robustness of clustering against adversarial noise. We design and evaluate two novel graph attacks against a state-of-the-art network-level, graph-based detection system. Our work highlights areas in adversarial machine learning that have not yet been addressed, specifically: graph-based clustering techniques, and a global feature space where realistic attackers without perfect knowledge must be accounted for (by the defenders) in order to be practical. To conclude the adversarial analysis, we propose two defense techniques that can improve the robustness of the clustering system.

We conclude the thesis in Chapter 6. Section 6.1 summarizes the contributions. In Section 6.2, we discuss additional attack cost for generalizing the adversarial analysis to other datasets as future work.

CHAPTER 2

BACKGROUND

2.1 Graph Clustering Methods

Many security datasets can be represented as graphs. Clustering is a common task performed to analyze these security graphs. The two basic assumptions employed by graph clustering techniques are homophily and structural equivalence. The homophily assumption is based on the notion that, "birds of a feather flock together". In other words, nodes that associate with each other are more alike. The corresponding clustering methods have been widely used in security applications. Community discovery identifies criminal networks [26], spectral clustering on graphs discovers botnet infrastructure [25], hierarchical clustering identifies similar malware samples [27, 28], and the associations in binary download graph can group potential malware download events [29, 30]. On the other hand, the structural equivalence assumption states that nodes with similar structural roles (e.g., sink nodes) should be in the same cluster, or, have similar graph embeddings. Newly devised graph embedding methods (e.g., DeepWalk [31], node2vec [32]) balance homophily and structural equivalence using node neighborhoods sampled by random walks. These methods could further improve the state of the art in the application of graph clustering in security research.

In this section, we explain graph clustering methods that will be used in the thesis.

2.1.1 Connected Component

In graph theory, within a connected component, any two vertices are connected, but there are no paths connecting the vertices that are in different components. Breadth-First Search and Depth-First Search algorithms can both compute the connected components of a graph.

We apply the connected component discovery in Chapter 4 to track malicious advertising campaigns.

2.1.2 Community Detection

There are many ways to detect communities in a graph. Several techniques in this space rely on a modularity metric to evaluate the quality of partitions, which measures the density of links inside and outside communities. This allows an algorithm to optimize modularity for community discovery communities. The Louvain algorithm [33] scales to large networks with hundreds of millions of vertices. Communities are usually hierarchical [34, 35, 36]; however, finding sub-communities within communities is a known hard problem [37]. This allows attackers to hide sub-communities in a "noisy" community by adding edges. We evaluate the community detection algorithm in adversarial settings in Chapter 5.

2.1.3 Spectral Methods

In [38], Braverman et al. discuss several popular spectral clustering strategies. First, a similarity matrix is used to represent the graph. Each row and each column represent a vertex to be clustered, and the weight is a similarity score between the corresponding vertices. After proper normalization, the matrix M is used as input to singular value decomposition (SVD) of rank k, $SVD_k(M) = U\Sigma V^*$. When the resulting eigenvectors (e.g., vectors in U) are further normalized, they can be used as an embedding in a euclidean space for learning tasks. In spectral methods, the hyperparameter k is usually chosen by first evaluating the scree plot of eigenvalues to identify the "elbow" where higher ranks have diminishing returns of representing the input matrix. When the scree plot starts to plateau at the ith eigenvalue, we set $k = i$ [39, 40].

Spectral clustering with SVD is known to have limitations when clusters are imbalanced; this is due to either graphs being scale-free (power law distribution) [41], or when small communities exist [42]. Unfortunately, both commonly occur in real-world data. In

practice, these small communities are merged into what is colloquially called the "death star" cluster: a large, noisy cluster that contains many small communities.

In Chapter 3, we propose a new spectral expansion algorithm, which uses spectral clustering, to analyze the impression fraud problem of the botnet TDSS/TDL4. In Chapter 5, we also analyze the robustness of spectral clustering in adversarial settings.

2.1.4 node2vec

Contrary to the strong homophily assumption of community detection and spectral clustering, node2vec [32] has the advantage of balancing homophily and structural equivalence in its embeddings. For example, vertices that are sink nodes will have similar embeddings. node2vec generates embeddings of vertices by optimizing the sum of the log likelihood of seeing the network neighborhood given a vertex v, for all vertices on the graph:

$$\max_{f} \sum \log P(N_S(v)|f(v)) \tag{2.1}$$

Where $f(v)$ is the embedding of vertex v, $N_S(v)$ represents the network neighborhoods of v with a series of vertices obtained by the sampling strategy S. node2vec proposes a sampling strategy by random walks starting from every vertex on the graph with the following parameters: 1) number of walks from each vertex, 2) length of each walk, 3) probability to return to the same vertex (Breadth First Search), and 4) probability to explore out to further vertices (Depth First Search). Once the walk samples have been obtained, node2vec uses a tunable *neighborhood size* to get the neighborhood of vertices. For example, a walk with length 5 $\{v1, v2, v3, v4, v5\}$ generates the following neighborhoods with size 3: $N(v_1) = \{v2, v3, v4\}$, $N(v_2) = \{v3, v4, v5\}$.

In order to compute the embeddings given $f(v)$, Equation 2.1 is factorized as a product of the conditional probability of each vertex in the neighborhood based on the conditional independence assumption. Each underlying conditional probability is defined as a sigmoid function, and the embeddings are learned by stochastic gradient descent (SGD) with neg-

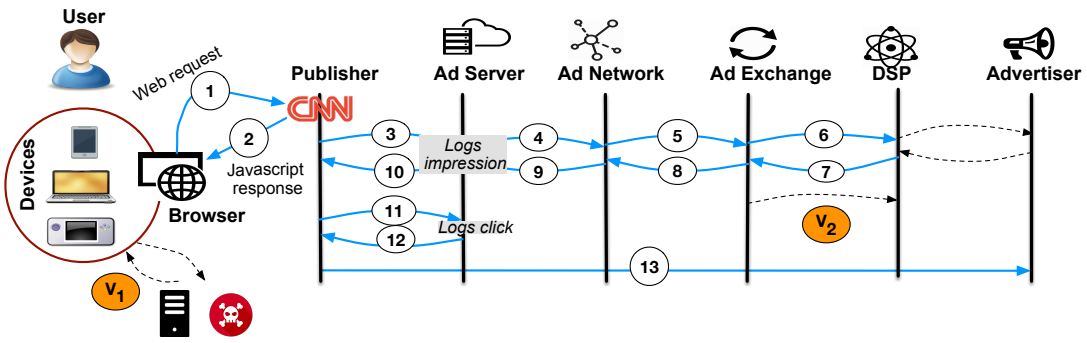


Figure 2.1: A brief overview of online advertising ecosystem.

ative sampling optimization. Effectively, node2vec learns embeddings in a fashion similar to word2vec [43] but does not use skip-grams. Attackers can target the neighborhood size and sampling parameters to encourage their vertices to be under-sampled and thus split into multiple noisy clusters. In Chapter 5, we evaluate node2vec in adversarial settings.

2.2 Online Advertising Ecosystem

Our study in Chapter 3 and Chapter 4 are from two different vantage points in the online advertising ecosystem. In Chapter 3, we are from vantage point V_1 (Figure 2.1), which is at the edge of the ad ecosystem, between infected machines and DNS/HTTP servers they connect to. In Chapter 4, we observe the ad bidding requests within the ad ecosystem at vantage point V_2, between ad exchanges and a DSP. To explain that, we describe the ad ecosystem in this section.

Figure 2.1 gives an overview of the online advertising ecosystem. There are three major components in the ad ecosystem: supplier, demander, and the marketplace. Suppliers are the publishers who provide *ad inventories* (e.g., spaces in the webpage), to show ads. Each time an ad is shown is called an *impression*. As "demanders", advertisers wish to secure the optimal ad inventories to reach the most relevant audiences. Various entities in the market place connect the demand the supply, which we use illustrate next.

When a user visits a publisher webpage (step 1, Figure 2.1), elements of the webpages are loaded (step 2), during which the iFrame representing the ad inventory of the webpage

requests from the ad server to display an ad (step 3). The ad server delivers the ad unit from an ad network (step 4), and also report any ad metrics available so a payment can take place. Each ad network has its own group of publishers, and it can also sell ad inventories to an ad exchange (step 5). If an ad request cannot be fulfilled, it will be further relayed to a Demand Side Platform provider (DSP) (step 6), and advertisers working with the DSP can purchase the impression. The advantage of using a DSP is that advertisers will have access to multiple ad exchanges. The advertisers can target users of a specific profile [44, 45], certain types of publishers, keywords [46], time of the day, flexible daily budget, etc.

The DSP, ad exchanges, and ad networks consolidates this information and shows the optimal ad back to the publisher's page (step 7 to 10). An impression is therefore fulfilled and logged. Impressions are often charged according the CPM (Cost Per Mille, or cost per thousand impression). If the ad is clicked, the ad server will log the click (step 11), and redirect the user (step 12) to the page of the advertiser (step 13). In such an event, the advertiser is charged by the click. The CPC (Cost Per Click) varies according to the keywords of the webpage and the user category.

Publishers can *syndicate* the ads to other downstream publishers. In turn, the syndicated publishers can *subsyndicate* the ads further to other publishers. Syndication enables the ads to reach a wider audience. Thus, there can be several redirections among publishers before ad request reaches ad server (step 3).

Omitted from Figure 2.1 are a number of quality control actors who interact with multiple phases of the impression flow. Such actors include ad verification companies (e.g., Integral Ad Sciences), fraud detection companies (e.g., ForensIQ), demographic verification companies (e.g., Nielsen), URL classification companies who provide context for topical ad selection (e.g., Peer39), and ad blocking companies (e.g., Ghostery).

Entities in the ad ecosystem perform fraud detection independently. The technical details are not disclosed in public documents in order to avoid evasion by the fraudsters [47, 48, 49]. As a countermeasure for fraud, ad networks employ smart pricing to normalize

CPC (Cost-Per-Click) based on distribution of conversion rates across all publishers [50, 48]. Different actions can be considered as conversion instead of a simple click or impression, such as product news subscription, purchase activity, filling out a questionnaire, etc. If traffic from a publisher results in a low conversion rate compared to other publishers serving similar ads, the ad network may use smart pricing to reduce the CPC used to calculate payment to that publisher. The drawback of the smart pricing policy is that advertisers have to share the conversion data with the ad networks. The conversion data are often considered sentitive information and therefore advertisers typically are not willing to share them. In practice, ad networks take many factors into account that would indicate the probability for a conversion [51]. No details about these factors are revealed. Nevertheless, since the conversion data are limited, attackers have been able to get payments based on CPC even after smart pricing discounts [52].

While smart pricing could make fraudulent clicks less profitable, this is not the case with fraudulent impressions. Only recently, Google and IAB announced the Ad "Viewability" standard in an effort to combat invalid impressions: at least 50% of ad pixels need to be in view for a minimum of one second [4, 3]. Advertisers can now choose whether to only bid on *viewable* impressions in the Real Time Bidding process [53]. However, it is still a nontrivial problem to correctly measure viewability.

CHAPTER 3

FINANCIAL LOWER BOUNDS OF ONLINE ADVERTISING ABUSE

3.1 Motivation

Many researchers have observed a shift in how botnets are monetized [54], away from traditional spam and bank fraud, towards advertising abuse [55]. Large botnets such as Kelihos [56] and Asprox have moved to monetization methods that abuse the online ad ecosystem. Unlike bank fraud and other types of abuse, impression and click fraud are "low risk/high reward" for botmasters, given the inherent difficulty in ad abuse attribution due to the complexity of the ad ecosystem [57].

To date, the evidence about the amount of ad-abuse attributed to modern botnets is sporadic, mainly because of measurement challenges. Studying the monetization components of botnets in a controlled environment (e.g., honeypots, dynamic malware analysis) requires researchers to *actively engage* in the abuse, which poses ethical challenges. In addition, dynamic malware analysis methods often fall short as botnets move their monetization components away from binaries [58, 52], and instead deliver them as separate, non-executable add-on modules. Such drawbacks point to the need for an efficient passive analysis system that can analyze the long-term *monetization campaign* separately from the traditional infection, Command and Control (C&C) and malware update methods.

To enable efficient, independent and passive analysis of the long-term ad-abuse caused by botnets, we introduce a novel Ad-abuse Analysis System (A^2S). A^2S leverages spectral clustering methods on passive DNS datasets to identify the network infrastructure (domain names and IP addresses) the botnet under inspection uses to perform ad-abuse. It also employs sinkhole datasets to estimate lower bounds of financial loss caused by the fraudulent impressions generated by the botnet.

Using four years of long-term network datasets, A^2S helped us estimate the scale of the ad-abuse potentially inflicted on advertisers from one of the most notorious botnets in history — TDSS/TDL4. Our conservative estimation shows that TDSS/TDL4 caused financial damage of *at least* $346 million in total; roughly $340 thousand per day. Furthermore, this estimate only includes less than 15% of the botnet's population, which suggests that the overall financial loss of advertisers caused by all bots is likely higher.

While these numbers may appear large, they remain an underestimation of the overall abuse due to the choices in our measurement methodology. We must emphasize that at every step of our analysis, we err on the side of being overly conservative, as we are interested in lower bounds. This helps us establish as conservative of a lower bound as possible, using aggressive, empirically driven filtering and relying on the lowest possible estimates for constants used in our financial abuse calculation. We intentionally exclude highly likely TDSS/TDL4 domains in exchange for a safer lower bound estimate.

We start by describing the necessary background information in Section 3.2. Next, we describe the details of our Ad-abuse Analysis System in Section 3.3. In Section 3.4, we describe the datasets used to evaluate the ad-abuse component of the TDSS/TDL4 botnet. We present the analysis of the botnet in Section 3.5, and two ad-abuse reports in Section 3.6. We discuss ground truth and accuracy of the analysis in Section 3.7. Related work are discussed in Section 3.8. We conclude with Section 3.9 and the takeaways from this Chapter.

3.2 Background

3.2.1 Botnets and Sinkholes

In the Domain Name System (DNS) [59, 60], domain names are composed of labels, separated by periods, which correspond to namespaces in a hierarchical tree structure. Each label is a node, and the root label (.) is root of the tree. The hierarchical concatenation of nodes creates a fully qualified domain name. A zone is a collection of nodes that constitute a subtree with *DNS authority name servers* responsible for its content. Figure 3.1 illustrates

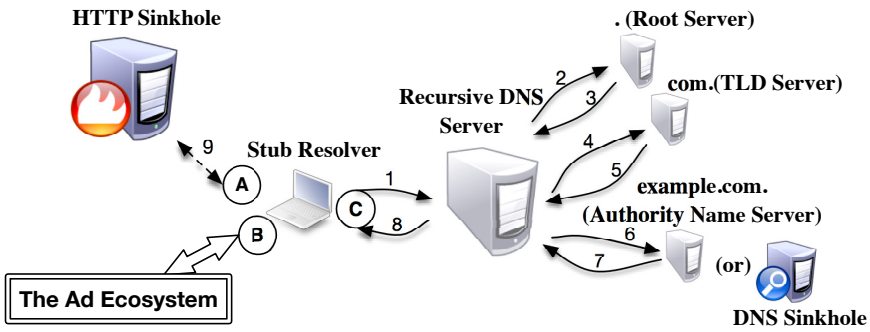


Figure 3.1: A high level overview of DNS resolution (1-8), the sinkholing processes (A) and the points where ad-abuse can be observed (B and C).

a typical resolution process. It begins with a stub resolver issuing a domain name resolution request for a domain, `example.com`, to the local recursive DNS server (RDNS) (see step 1, Figure 3.1). In the event that the RDNS does not have the resolution answer in its cache, it will begin an iterative process to discover it. The RDNS will iteratively "walk" the DNS hierarchy, starting from root server (steps 2 and 3), to the next level of effective top-level domain (TLD) server (steps 4 and 5), and down to the authority name server (ANS) for the requested zone (steps 6). Once the RDNS receives (step 7) the authoritative mapping between the requested domain names and its corresponding answer (e.g., IP address) from the authority, it forwards the answer back to the stub resolver (step 8).

After a command and control (C&C) domain for a botnet is resolved, the next step is a connection attempt (e.g., HTTP GET) from the stub to the C&C server. Network administrators and security researchers often take over such C&C domain names to change their DNS setting, effectively making them point to a new location. This is commonly known as "sinkholing" a domain name [61]. If `example.com` is sinkholed, the stub resolver will establish any future C&C connections to the sinkhole (step 9, Figure 3.1) rather than the adversary's C&C server.

In addition to sinkholing a domain's A/AAAA record, one can also sinkhole the ANS that serves it. For instance, `example.com` can be sinkholed by changing the ANS record

to a server under the control of the sinkholing party (e.g., law enforcement or security researchers). Such an action would have the following result: during the DNS lookup chain in Figure 3.1, after steps 1 to 5, the recursive DNS server will ask the new DNS sinkhole server controlled by the sinkholing party about the authoritative answer for the domain name. Sinkholing both the domain name and the ANS server is a common practice in the security community as it provides telemetry from both the DNS resolution and network communication planes of the threat being sinkholed.

Attackers often change C&C domains to avoid sinkholing. Domain name generation algorithms (DGAs) [62, 52] can be used to rapidly update the C&C domains to remain agile against sinkholing efforts. A DGA can be implemented client-side in the malware sample itself, or server-side in the C&C server. Intuitively, client-side DGAs can be reverse engineered from the malware sample. Unfortunately, server-side DGAs are much more difficult to understand since the server computes and pushes new C&C domain configurations to the bots. Reverse engineering requires obtaining the C&C server code, which is often heavily protected by the author. However, monitoring traffic from infected hosts guarantees the observation of C&C domain changes.

3.2.2 Observing Ad-abuse In Local Networks

To understand where and what a network administrator can monitor, we need to examine the typical life cycle of an infected host. First, the malware looks up the IP address of the C&C domain (point C in Figure 3.1). Second, it contacts the C&C server to get commands for doing impression and click fraud (point A in Figure 3.1). Next, the malware attempts to execute the commands by interacting with the ad ecosystem (point B in Figure 3.1). Stealthy malware carries out these tasks by blending in with users' normal web browsing activities in order to evade anti-abuse detection within the ad ecosystem. Additionally, the malware often reports back to the botmaster various byproducts from the monetization activities (e.g., user's search history) for bookkeeping of the monetization campaign.

Typical egress monitoring functionality can be used to observe different aspects of ad-abuse. Administrators can observe interaction between infected hosts and the Internet infrastructure that supports the monetization campaign (points A, C), or between infected hosts and the ad ecosystem (point B in Figure 3.1). From the network's point of view, this observation takes the form of DNS resolutions (i.e., for the domains facilitating ad-abuse from point C in Figure 3.1) and any application-layer C&C communications between local victims and the ad ecosystem (point B in Figure 3.1). We select observation points A and C in Figure 3.1, so we can mine sinkhole and DNS datasets. Points A and C correspond to the vantage point V_1 in Figure 2.1, Section 2.2. We should also note that HTTP connections can be observed for the sinkholed domain names (point A in Figure 3.1). The sinkholing party did not return any commands to bots. Therefore, the communications to the sinkhole did not, at any point, reach the ad ecosystem. This means that our efforts to study the botnet did not contribute any additional abuse to the advertisers and other parts of the online advertising ecosystem.

3.3 Ad-abuse Analysis System

In this section we introduce the Ad-abuse Analysis System (A^2S, Figure 3.2) that allows administrators to systematically analyze ad-abuse in their networks. The goal of the system is to provide a detailed analysis of the Internet infrastructure that supports ad-abuse monetization. Such information helps administrators to *independently* (1) estimate the level of ad-abuse that victims in the local networks contributed to the entire ad ecosystem and (2) obtain a set of domain names and IPs that can be used for network policy actions. Network administrators can take action against the monetization component of the botnet. If adversaries cannot monetize infected hosts in a network, the hosts in the network becomes less appealing to compromise.

The system consists of three logical components: (i) the necessary datasets for its operation, (ii) a module to link sinkholed network traffic with long-term passively collected

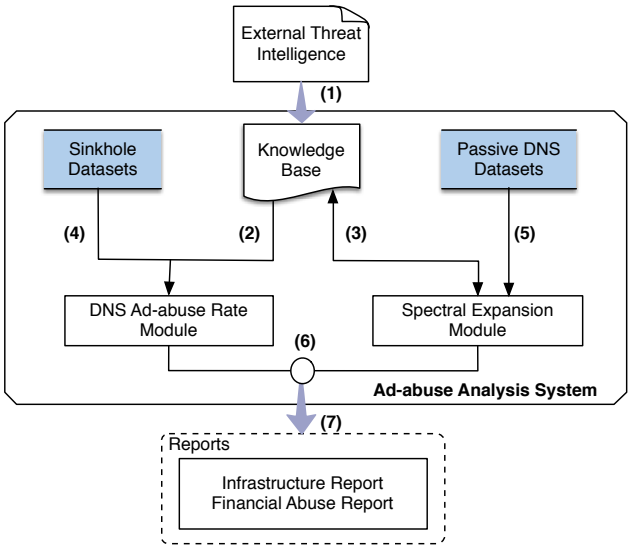


Figure 3.2: Overview of the Ad-abuse Analysis System (A^2S).

DNS datasets, and (iii) a module to identify additional ad-abuse domains using passive datasets. We begin by providing an overview of A^2S.

3.3.1 System Overview

The first input of A^2S is ground truth C&C domains obtained by either external threat reports or manual analysis of a particular threat (Step (1), Figure 3.2). These reports are added to our knowledge base, and act as input for two different modules: the DNS Ad-abuse Rate Module (Step (2)) and the Spectral Expansion Module (Step (3)).

The **DNS Ad-abuse Rate Module** estimates how many ad-abuse events, i.e., C&C connections requesting for impression or click fraud commands, are typically triggered after a single DNS resolution request for any ad-abuse domain (Step (4)). This can be achieved by "taking-over" a small portion of such ad-abuse C&C domain names for a period of time. The takeover can be done by traditional sinkhole methods or commonly used walled garden policy techniques [63] at the recursive DNS level and perimeter egress points of a network.

The **Spectral Expansion Module** identifies a set of domain names that have been used

by the ad-abuse campaign *historically*. We assume that infected hosts contact both known and unknown ad-abuse domains and these domains share Internet infrastructure. We combine ground truth from external threat intelligence with large passive DNS datasets (Step (5)). The passive DNS datasets enable the creation of a graph between the bots in the local network and the Internet infrastructure contacted by the bots. The graph is represented by an association matrix. Spectral analysis of the association matrix enables us to extend the ad-abuse domains to a larger set that is highly related to the ground truth. The module iteratively expands the set of ad-abuse domains using sliding temporal windows and improves our understanding of the long-term ad-abuse operation (Step (3)). After expansion, the module sanitizes extended ad-abuse domains using historical WHOIS information to eliminate false positives.

The resulting output from both modules will be combined (Step (6)) to derive the final reports (Step (7)). The infrastructure report includes all domain names and IP addresses used by the ad-abuse campaign. These domains and their historical DNS lookup volumes are used to generate the financial abuse report. We use a financial model to approximate a lower bound of advertisers' loss caused by the campaign.

3.3.2 Datasets to Study Ad-abuse

Before we describe the two modules in A^2S in detail, we need to explain the necessary datasets required to analyze an ad-abuse campaign. These datasets include sinkholed traffic from ad-abuse domain take-over actions and passive DNS datasets that contain historical DNS resolutions for domain names observed in the local network.

Following a DNS query, the infected host will request commands from the C&C server. For instance, the infected host could request a list of ads to view or click on, report user content such as cookies and recent search terms, or even report fraudulent clicks and impressions that took place. These communications between the infected host and the ad-abuse C&C servers can be observed in the sinkhole datasets, which should include both the

DNS resolution requests for sinkholed domain names and all application layer (i.e., HTTP) communication attempts towards the sinkhole infrastructure.

In addition to the sinkhole datasets, A^2S needs passive DNS datasets. Specifically, the datasets are recursive DNS query traffic (qDNS) from the hosts within the network and traditional deduplicated passive DNS datasets (pDNS-DB) that record domain names and their historical resolutions. A^2S uses these datasets to identify ad-abuse C&C domains from ground truth set $D_\$$ to a bigger set D_A, which we will thoroughly explain in Section 3.3.4.

Often, ad-abuse botnet modules use domain name generation algorithms (DGA) [62, 52] to facilitate the fraudulent activities. In such events, we can use DNS queries that result in "non-existent domain" (a.k.a. NXDOMAINs) as the qDNS dataset since unknown C&C domains may have never resolved. The use of DGAs is the most complicated case of ad-abuse. TDSS/TDL4 uses a server-side DGA to facilitate ad-abuse.

3.3.3 DNS Ad-abuse Rate Module

The DNS Ad-abuse Rate module quantifies the number of ad-abuse events that are performed after a single DNS request. In this case, the ad-abuse events are the C&C connections asking for impression or click fraud commands. This ad-abuse rate maps DNS lookup volume to the number of total ad-abuse events. To properly compute the rate, the module needs to analyze DNS queries and application-layer HTTP requests to sinkholed ad-abuse domains.

We define the "DNS Ad-abuse Rate" as $\zeta = y/x$, where x is the number of domain name resolution requests for the sinkholed domains and y is the number of application-layer communication attempts that reflect ad-abuse events. In other words, the module needs to observe x domain name resolution requests and y HTTP connections to the sinkhole, within a time window t, to safely assume a ζ level of ad-abuse happened with each historical ad-abuse domain lookup. Administrators can collect such sinkhole datasets either by acquiring

a commercial sinkhole data feed or by independently taking over the ad-abuse domains, locally or globally.

Using ζ, the module can provide the system the ability to *pivot* from "short-term" sinkhole observations to "long-term" passive DNS observations. Specifically, we can use the DNS Ad-abuse Rate to analyze many years of DNS traffic related to the ad-abuse operation using passive DNS datasets. We now discuss how A^2S mines these datasets.

3.3.4 Spectral Expansion Module

The Spectral Expansion module uses local network traffic to reason about the domain names used for the ad-abuse operation, over a long period of time. The module accurately identifies additional domains based on original ground truth knowledge of the ad-abuse operation, using a large passive DNS dataset. In other words, the spectral expansion module is able to take a set of ground truth domains, $D_\$$, and eventually derive a larger set of domains, D_A, that have historically participated in the ad-abuse activities.

A^2S derives D_A using spectral clustering on DNS datasets from the local network. The spectral expansion algorithm iterates through the entire DNS query dataset (qDNS). Each iteration walks over DNS data for a given day, with the goal of discovering new ad-abuse domains that will be added to the D_A set.

We conservatively assume that unknown ad-abuse domains were queried by a common group of infected hosts, or they pointed to the same Internet infrastructure that served the known ad-abuse domains over the same temporal window. Each day, we create a tripartite graph that "links" candidate domain names, their resolved IP addresses or Canonical Names (CNAMEs), and the network hosts that queried for them. The association matrix representing such a graph can be seen in Figure 3.3.

Spectral decomposition of this matrix enables this module to group candidate domain names that either share common Internet infrastructure and/or local network hosts that queried them, via standard clustering methods. Then we analyze the clusters to add domain

Algorithm 1 Spectral Expansion Algorithm

Require:δ

1: $H \leftarrow \{h | \exists q \in D_A$: h queried q on day $d_i\}$
2: $D \leftarrow \{q | \exists h \in H$: h queried q on $d_i\}$
3: $Rdata \leftarrow \{ip | \exists q \in D$: q resolved to ip historically$\} \cup \{cname | \exists q \in D$: q resolved to $cname$ historically$\}$
4: Apply thresholds α and β to the sets of $Rdata$ and H, respectively, to remove noisy IPs and hosts.
5: $M \leftarrow$ relationship between D and ($Rdata$, H). Normalize by IPs, CNAMEs and Hosts.
6: $S \leftarrow M \times M^T$
7: $U \Sigma V^* \leftarrow SVD(S)$
8: $clusters \leftarrow XMeans(U)$
9: $D_A \leftarrow$ Analyze $clusters$.
10: $i = i + \delta$, Go to line 1.

names to D_A. Domains are added if they have explicit relationships with already known ad-abuse Internet infrastructure or share common infected hosts.

Algorithm 1 formally describes the spectral expansion process. Each iteration of the algorithm processes the DNS resolutions of day, d_i, to update the ad-abuse domain set, D_A. The operator can set δ to determine how the algorithm iterates through time.

Next we discuss the steps in detail for one iteration. Initially we assume that $D_A = D_\$$. The first four steps prepare necessary data for assumbling the association matrix for domains of interest. In the **first step**, the algorithm identifies all internal network hosts (H) querying any known ad-abuse domain in D_A. In the **second step**, the algorithm narrows down potential unknown ad-abuse domains to all domains (D) queried by infected hosts (H). In the **third step**, we obtain all historical IP addresses and CNAMEs for domain names in D from the local passive DNS database, denoted as $Rdata$.

During the **fourth step**, the algorithm removes any "noisy IP addresses" from $Rdata$ and "noisy hosts" from H. IP addresses that are likely used for parking or sinkholing and hosts that are probably large gateways or part of security research infrastructure can introduce noisy association between domains that do not reflect ad-abuse behavior (see Figure 3.3). The algorithm excludes such "noisy" IPs and hosts by using two aggressive

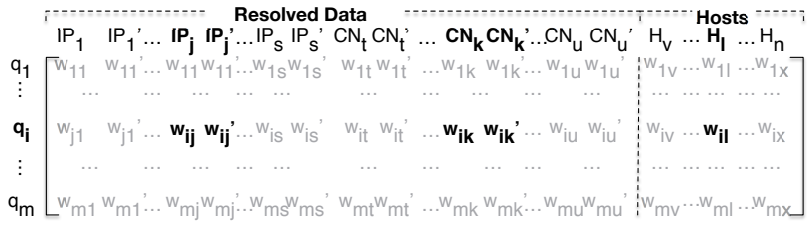


Figure 3.3: Association matrix for domain, RDATA, and host.

thresholds. Note that aggressively removing noisy IP addresses and hosts reduces connections in the graph that could have led to more C&C domains, but results in safer estimation that is still a lower-bound.

The first threshold (α) denotes the number of related historical domain names for an IP address seen from network traffic on the local network. We exclude IPs with an unusually high number of domains. The second threshold (β) relates to the number of domains queried by an infected host. In this case, if the number of domains queried by a host is more than what's typical for infected hosts in the local network, we exclude it from the set H. The way we reason and select the actual values of α and β will be discussed in Section 3.5.2.

In the **fifth step**, the algorithm builds an association matrix linking the domains in D with the set of IP addresses and CNAMEs in $Rdata$ and the set of internal hosts in H that queried them. An example matrix is shown in Figure 3.3. The rows represent all domains queried by infected hosts ($q_1 ... q_m$), and the columns reflect historically resolved IPs/CNAMEs and the hosts that queried those domains in the day d_i. In order to assemble the matrix we compute two types of weights.

The first weight represents the DNS lookup properties from the domains in $Rdata$, with respect to IPs and CNAMEs. Specifically, the weights w_{ij} and $w_{ij}^{'}$ are the timestamps for the first day (w_{ij}) and the last day ($w_{ij}^{'}$) we observed domain name q_i resolving to IP_j. In the same sense, the weights w_{ik} and $w_{ik}^{'}$ are the timestamps for the first and last day we observed domain name q_i resolving to CNAME CN_k.

The second weight represents a binary indicator of whether the particular domain name in D was queried in day d_i by an internal host in H. Specifically, if host $host_l$ queried domain q_i on day d_i, the weight value w_{il} equals 1; otherwise, w_{il} equals 0. After the matrix has been assembled, the algorithm will normalize by row (for each q_i) the sum of "IP" values to one, the sum of "CNAME" values to one, and the sum of "Host" values to one.

In **step six** the algorithm transforms the association matrix $M_{m\times n}$ to its corresponding similarity matrix $S_{m\times m}$. This matrix represents how similar domain name q_i is to any other domain q_j. During the **seventh step**, the algorithm performs Singular Value Decomposition (SVD) on S, and obtains $U\Sigma V^* = SVD(S)$. The first twenty eigenvalues are kept for **step eight**, where the twenty-dimensional eigenvectors are clustered by XMeans [64].

Step nine analyzes the resulting clusters and finds new ad-abuse domain names. This cluster characterization process propagates the existing labels from ad-abuse domains in our knowledge base to unknown domains. The label propagation rules are based on both IP infrastructure overlap and querying host overlap between domains. We discuss how we propagate these labels based on cluster specific thresholds in Section 3.5.2. The known ad-abuse domain names set D_A is updated with the newly discovered domains.

The **tenth** and final step of the algorithm restarts the algorithm from the first step. Depending on the value δ set by the administrator, the algorithm determines the day to check next; for $\delta = 1$, the algorithm proceeds to the next day, whereas $\delta = -1$ forces it to go backwards in time. This is very useful when the original ground truth domains were seen in the middle of the long-term network observations. Using the updated set D_A, the system can identify more ad-abuse domains. After reaching the last day of available data according to the iterating direction specified by δ, the algorithm stops.

Finally, the module sanitizes the derived D_A to exclude mistakenly characterized ad-abuse domains based on historical WHOIS information. We extract email addresses and name servers from WHOIS for each domain in D_A, and compare these with known emails

and name servers used for the ad-abuse domains in $D_{\$}$. If either email or name server matches, the newly discovered domain is kept in D_A. Otherwise, we exclude the domain from financial analysis. Thus, the derived D_A will be used to estimate conservative lower bounds of ad-abuse in the local network.

3.3.5 Reports On Ad-abuse And Financial Models

Outputs from the DNS Ad-abuse Rate and Spectral Expansion Modules are combined with further analysis of pDNS-DB to generate two reports. The first report describes the network infrastructure used to facilitate the ad-abuse, using historical IP addresses derived from the extended ad-abuse domains D_A. These domains, along with the DNS Ad-abuse Rate and the daily DNS lookup volumes, are used to generate the second report that estimates the daily and overall financial impact of ad-abuse to the online advertising ecosystem.

To derive the financial model used to calculate the abuse M to the ad-ecosystem, we first consider advertisers' loss on both fraudulent clicks and impressions in the generic case described by the following equation:

$$M_{generic} = \sum_i \zeta * R_i * (p_{clk} * \mu_{clk} * CPC + p_{im} * \frac{\mu_{im}}{1000} * CPM) \tag{3.1}$$

For each day i, advertisers' loss is calculated based on the number of DNS requests R_i to $d \in D_A$ observed in the local network. $\zeta * R_i$ reflects the total number of ad-abuse HTTP connections for C&C purposes. We separate the connections in $\zeta * R_i$ into two different components. The p_{clk} component reflects the percentage of HTTP connections for click fraud communications. The second component p_{im} represents the remaining percentage of HTTP connections that corresponds to impression fraud communications. Since each connection may contain multiple clicks or impressions, μ_{clk} and μ_{im} represents the multiplicative factor for the model to derive the total number of clicks or impressions, respectively. The number of clicks multiplied by the CPC (Cost Per Click) allows us to compute the click-fraud abuse, while the number of thousand impressions multiplied by

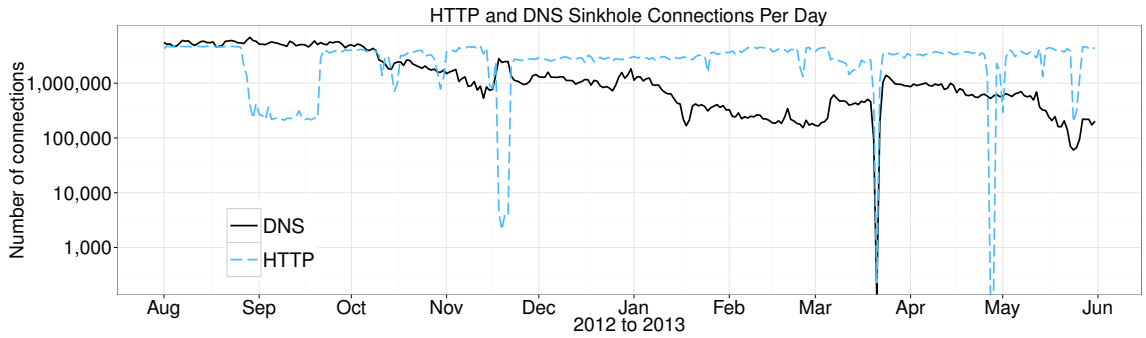


Figure 3.4: Number of requests received by the DNS and HTTP sinkholes over 10 months.

the CPM (cost-per-thousand impressions) allows us to calculate the financial loss from the fraudulent impressions. Finally, the sum of financial loss from all days in the dataset is the total loss that the advertisers endured due to the infections in the local network.

Equation (3.1) shows advertisers' loss if no fraud was detected by entities in the ad ecosystem. Since we want to derive the lower bound of advertisers' loss, we assume that the botnet under inspection did not profit from fraudulent clicks, due to smart pricing policies (effectively setting $CPC = \$0$). In addition, we assume that all impressions related ad-abuse was successful, since the Ad "Viewability" standard has only recently seen some traction [4, 3]. The derived financial model for impression is the following:

$$M_{impression} = \sum_{i} \zeta * R_i * (p_{im} * \frac{\mu_{im}}{1000} * CPM) \tag{3.2}$$

We would like to emphasize that with model $M_{impression}$ we assume that smart pricing policies were perfect across the entire ecosystem and no click fraud made profit at any point in the lifetime of the botnet operation. The financial model $M_{impression}$ assumes that the attackers were able to monetize fraudulent impressions from infected hosts. This is a realistic assumption since detecting impression fraud has been extremely challenging to date [65, 54]. We caution the reader that fraudulent clicks could still be successfully monetized even after smart pricing normalization, although the CPC may be reduced to a small percentage of the standard CPC [7]. To precisely estimate this percentage it would require us to obtain data from several affected entities in the ad ecosystem over the lifetime

Table 3.1: Summary of datasets.

	Date Range	Size	Records (millions)
DNS Sinkhole	8/1/2012 - 5/31/2013	6.9G	565
HTTP Sinkhole	8/1/2012 - 5/31/2013	248.6G	919
NXDOMAIN	6/27/2010 - 9/15/2014	133.5G	13,557
pDNS-DB	1/1/2011 - 11/6/2014	17.9T	10,209

of the botnet. Since we are interested in lower bounds, we decided to simply use the conservative choice of $M_{impression}$ as the financial model that would help us generate the ad-abuse financial report for the botnet.

3.4 Dataset Collection

In order to increase the situational awareness behind the problem of long-term ad abuse, we decided to analyze the ad-abuse component of the TDSS/TDL4 botnet, one of the most sophisticated, complex, and long-lived botnets in history. The ad-abuse component is agile because it uses a server-side DGA to generate its C&C domains. As our "local network" we selected one of the largest US Internet Service Providers (ISP). The ISP provided over four years of historical network data, permitting testing of A^2S on a large scale. Table 4.1 summarizes the datasets.

3.4.1 Sinkhole Datasets

We obtained sinkhole DNS and HTTP traces for the ad-abuse component of TDSS/TDL4 from two security companies. The goal of obtaining these datasets is to quantify the DNS Ad-abuse Rate (Section 3.3.3). To do so, we need to understand the type of HTTP connections in the datasets.

The datasets span over 10 months, during which the Authoritative Name Server (ANS) and the application-layer (HTTP) sinkhole points experienced some sporadic data loss due to collection problems and outages. We should note that all domain names that were sink-

holed had a zero time-to-live (TTL) value, which prevented caching at the recursive DNS server level, forcing it to contact the DNS sinkhole server for every lookup. Moreover, the HTTP sinkhole returned "HTTP 200 OK" answers back to the victims with no content. That is, the sinkhole administrator did not actively engage in ad-abuse.

Figure 3.4 shows the number of requests received by the DNS and HTTP sinkholes over 10 months. The downward spikes indicate data loss events. In the first two months, the volume of HTTP requests is lower than that of DNS resolutions, which we suspect is due to data collection issues. Starting from the middle of October, the HTTP requests out-numbered the DNS resolutions, as expected.

TDSS/TDL4 uses two C&C protocols for its ad-abuse operation. Both protocols were present in the HTTP datasets we obtained. The first protocol, "Protocol 1", is the primary mechanism through which the botnet performs impression fraud. This is achieved via an HTTP GET request to the active C&C, which will reply back with a set of advertisement URLs used for impression fraud. Among other information, Protocol 1 also reports the version of the malware and a unique identifier for each victim, namely *bid*. All these observations are in-line with data collected and analyzed by other security researchers [66, 67]. The second protocol, "Protocol 2", is used to report back information including search terms from the victim's browser, the publisher's website where ads have been *replaced* and *clicked on*, and the original ad that was replaced from the publisher's website. A semantically similar behavior of TDSS/TDL4 botnet is identified by Vacha et al. [68], where fraudulent clicks were only generated when a user engaged in real clicks. In order to protect infected users' privacy, the search terms were given to us in an aggregated form such that they cannot be mapped to the individual ID and the infected IP.

In total, we observed 565 million unique DNS resolution requests. 544 million requests were Protocol 1 and 21 million were Protocol 2 connections. This traffic was produced by 47,525 different recursive DNS servers (RDNS) around the world. Hosts with 66,669 unique identifiers (ID) contacted the HTTP sinkhole, using 615,926 different IP addresses.

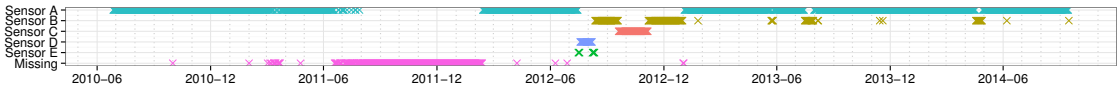


Figure 3.5: Sensor availability for the NXDOMAIN dataset over four years. 247 out of 1,542 days are missing.

They made 343 million unique `HTTP GET` requests using properly formatted base64 encoded URLs. 919 million connections were recorded, only 0.87% of which were Protocol 2 communication, while the rest 99.13% were Protocol 1 connections. Thus, we assigned $p_{im} = 99.13\%$ for Equation (3.2).

3.4.2 Passive DNS Datasets

We gathered two types of DNS datasets from a large US ISP that represents approximately 30% of DNS traffic in the US. The first is the NXDOMAIN dataset, which covers over four years of DNS queries from clients of the ISP for domains that did not resolve at the time of query. The second dataset we obtained is a historical passive DNS database (pDNS-DB), from the same ISP, containing DNS resource records (RR) [59, 60] collected from 1/1/2011 to 11/5/2014.

The NXDOMAIN dataset was collected below the recursive DNS servers, capturing queries from hosts to the recursive DNS servers that result in DNS answers with a return code of "NXDOMAIN". Throughout the four-year period, we gained access to 1,295 days of NXDOMAIN data (qDNS) from the ISP sensors (Figure 3.5).

The pDNS-DB dataset contains over 10 billion RRs. Each RR provides resolved data and the daily lookup volume of a queried domain name. The pDNS-DB was collected from 24 geographically diverse ISP collection points in the United States.

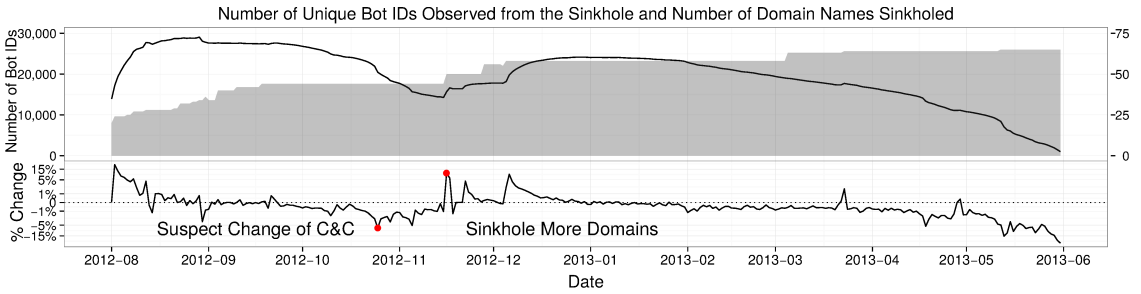


Figure 3.6: Top: The line plot shows victim population of the botnet sample that contacted the sinkhole infrastructure, with y-axis on the left. The area plot shows the number of sinkholed domains with y-axis on the right. Bottom: Percent change.

3.5 Analysis and Measurements

In this section, we discuss how we compute the DNS Ad-abuse Rate, and how we propagate ad-abuse domains from ground truth $D_{\$}$ to the larger set D_A for TDSS/TDL4.

3.5.1 Computing the DNS Ad-abuse Rate

Since we did not operate the sinkholes, we need to make sure that the datasets we obtained are generic enough and not biased before we can compute the DNS Ad-abuse Rate for the TDSS/TDL4 botnet. Thus, we will first summarize the sinkhole datasets to make the DNS Ad-abuse Rate reproducible by other researchers.

First, we need to understand the average lifetime of an infection using the unique infection ID. Each connection to the HTTP sinkhole contained the victim's IP address and a unique victim identifier. This identifier was a 40-byte long hexadecimal value that was tagged by TDSS/TDL4 malware as *bid* in Protocol 1 communications. Figure 3.7a shows the cumulative distribution function of the *average infection duration* based on IP address and victim ID. The results show a relatively longer infection lifetime for the victims when we count them using the unique identifier than when we use the victim's IP address. Counting bots by IDs is more accurate than counting by IP addresses due to Network Address Translation (NAT) points and DHCP churn rates, as other researchers have already noted [69].

Second, we want to examine whether the sinkhole traffic can cover a large victim population, through which we can safely generalize the ad-abuse observation. The daily victim population can be measured by the number of unique daily IDs that contacted the sinkhole. Figure 3.6 illustrates how the number of daily victims changes over time and the percentage of change [70] for the botnet observed from the sinkhole data. In the first two months of the datasets, the number of infected IDs reached a maximum of almost 30,000. After a sudden 6.7% drop in October, the number of IDs seen daily in our datasets decreased, until the middle of November 2012. The decrease indicates that the malware changed C&C domains from sinkholed domains to others. At that point the sinkhole administrators "refreshed" the sinkhole by adding six new domain names for the same botnet. This caused an increase in the number of IDs that were found in the sinkhole datasets. It is worth noting that a large number of old IDs reappeared in the sinkhole data after the addition of these six new domains. This observation is expected, as the server side DGA churns through new domains and old infections catch up with the new sinkholed domain names. After a peak of almost 8.9% increase at the end of 2012, the daily victim population remained around 23,000 until the middle of February 2013. Afterwards, the size decreased by a factor of almost 2% daily.

Finally, we need to examine the geographic distribution of the infected population. As our passive DNS datasets were collected at a US ISP, we want to make sure that the sinkhole dataset contains a reasonable size of victims located in the US. We identified the corresponding CIDR and Autonomous System Number (ASN) for each victim IP address [71], and used historical data from Regional Internet Registries (RIR) to find the country codes for the identified ASNs. Table 3.5a shows that almost half of the sinkhole traffic originates from victims in the US (46.77%). In total, 174 countries were affected, however, only 15,802 infections resided in countries outside the top six. These results show that TDSS/TDL4 traffic in our pDNS-DB dataset will allow us to study less than 15% of the entire botnet. This is due to the fact that the passive DNS dataset is collected from an ISP

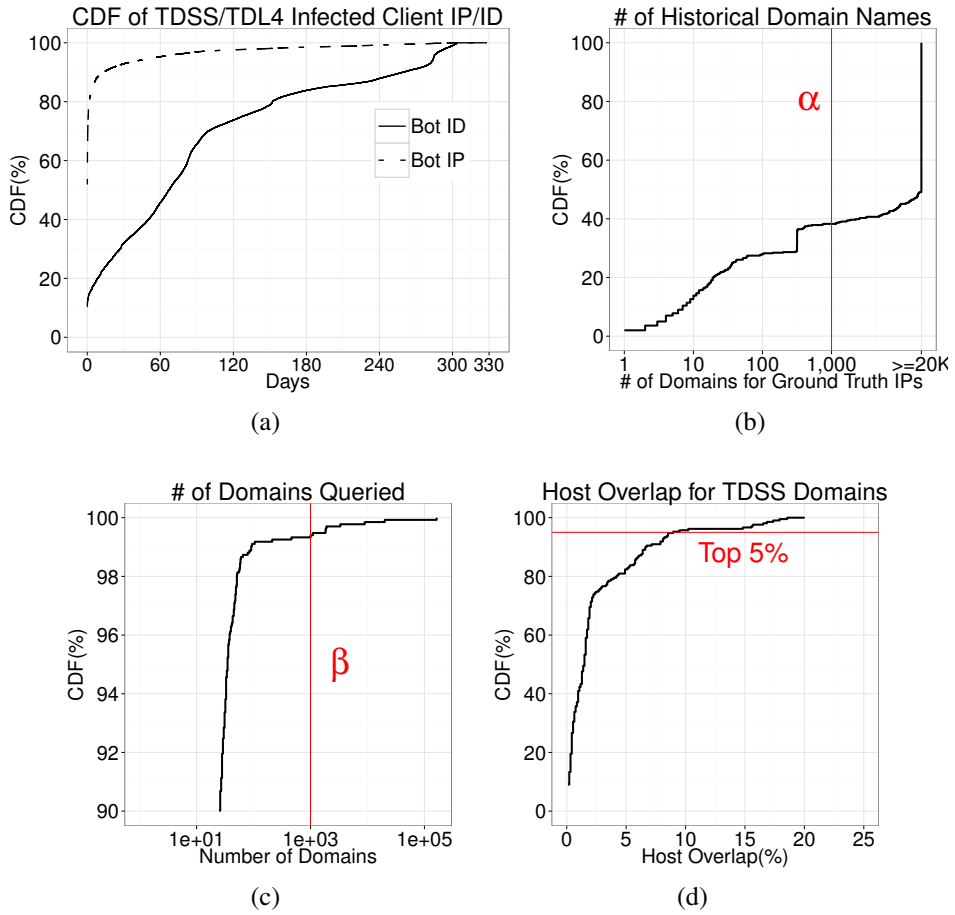


Figure 3.7: 3.7a: Cumulative distribution function (CDF) for the infection duration based on the infection ID and IP address. 3.7b: CDF for number of related historical domain names per IP from initial ground truth ($D_{\$}$). 3.7c: CDF for the number of domains queried by internal hosts (H). 3.7d: CDF for host overlaps for TDSS/TDL4 ground truth domains.

in the United States, which represents 30% of the overall DNS traffic in the US.

Computing the DNS Ad-abuse Rate ζ: Since our pDNS-DB dataset was obtained from a US ISP, we calculated the DNS Ad-abuse Rate ζ^{USISP} based on the sinkhole traffic that reflected victims in the particular ISP. This resulted in 9,664 unique victim IDs, 28,779,830 DNS connections, 154,634,443 HTTP Protocol 1 connections and 1,159,027 HTTP Protocol 2 connections over an observation window of 10 months. Using this ISP-specific dataset, we can compute the daily DNS Ad-abuse Rate and get the mean for the entire ISP as $\zeta^{USISP}_{mean} = 27.62$. We used $\zeta^{USISP}_{mean} = 27.62$ as the final DNS Ad-abuse Rate for our experiments. As discussed in Section 3.4.1, DNS caching will not bias our rate,

since the sinkhole administrators set a TTL equal to zero for the domains they sinkholed.

3.5.2 Spectral Analysis

Utilizing both NXDOMAIN and pDNS-DB datasets, we identified additional ad-abuse domains starting from our limited ground truth ($D_\$$) and ending up to a larger set (D_A) that supported the ad-abuse component of the TDSS/TDL4 botnet over four years. We derived this new set of domains D_A by using Algorithm 1 described in Section 3.3.4. In this section we discuss the operational challenges we faced while running this algorithm and how we managed to ensure the accuracy of the derived set D_A.

Assembling the Association Matrix

Algorithm 1 employs spectral clustering methods, which require a sparse association matrix as input. A^2S builds a matrix between domains we want to cluster as rows. As columns, we combine all historical RDATA of the domains and the infected hosts that queried them. Spectral clustering of such matrix results in clusters of domains that shared network infrastructure over the same time or were linked by infected host(s).

Before we constructed the association matrix (see Figure 3.3), we removed noisy IPs and internal hosts from the sets $Rdata$ and H based on the thresholds α and β. These thresholds were chosen because they reflected extreme cases of IPs and internal hosts, with respect to the local network.

Threshold (α) for Noisy IPs: Figure 3.7b shows the number of historical domain names per IP address, which were manually labeled from the TDSS/TDL4 ad-abuse domains in $D_\$$. Under 40% of confirmed TDSS/TDL4 C&C IPs historically have fewer than 1,000 domains pointing to them. Concurrently, over 50% of these IPs have more than 20,000 related historical domain names. Such IPs are likely used for parking or sinkholing purposes. We manually analyzed the set of IP addresses with around 1,000 related historical domains to assess whether they are malicious. The analysis revealed that considering

IPs with more than 1,000 historical domains as noisy is an aggressive threshold. However, since we are estimating the lower-bound of TDSS/TDL4 ad-abuse operation, falsely removing IPs that were not used for parking or sinkholing will only help our lower bound goal. That is, such aggressive threshold will only remove links within the association matrix that would have allowed us to discover additional ad-abuse domains that could be added to the set D_A.

Threshold (β) for Noisy Hosts: Figure 3.7c shows the cumulative distribution of the number of domains queried by infected hosts in a day. Note that the x-axis is in log scale and the y-axis starts at 90%. The plot shows that only 0.7% of infected hosts queried more than 1,000 domain names in a day. These hosts are likely gateways or research infrastructure that don't necessarily associate known with unknown ad-abuse domains during the clustering process. Thus, we used the 1,000 mark as a threshold. This means that any host that queried more than 1,000 domains in a day was instantly excluded. Again, this is an aggressive threshold, which rather forces us to underestimate the number of infected hosts (and yield again to lower bounds).

Using these thresholds, we constructed the sparse matrix, performed Singular Value Decomposition, and extracted the first 20 eigenvalues, which we used to cluster the domains in the matrix using XMeans [64].

Cluster Analysis

After clustering, we labeled ad-abuse domains based on IP infrastructure and infected hosts.

IP infrastructure: From clusters containing known ad-abuse domains, we label other unknown domains as ad-abuse domains if they share the same IP infrastructure. This provides a subset of all domains pointing to the IP infrastructure used by TDSS/TDL4, since domains in these clusters have been queried by infected hosts and have resolved during the same time frame as known ad-abuse domains.

Internal (Infected) hosts: Since TDSS/TDL4 uses a server-side DGA, unknown C&C

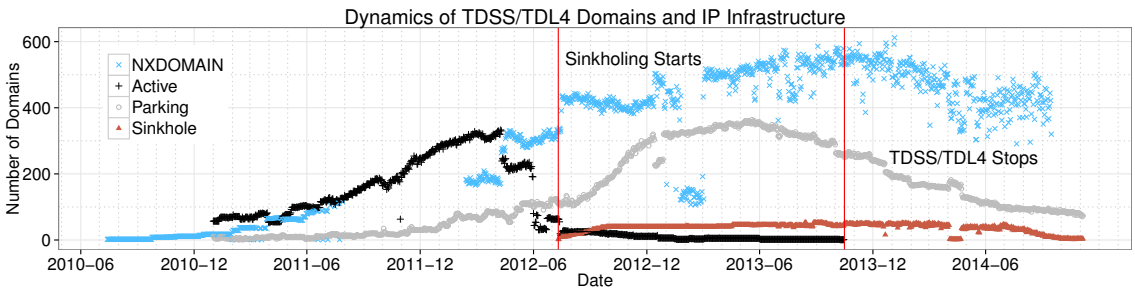


Figure 3.8: Evolution of TDSS/TDL4 domains and their IP infrastructure. The number of active domain names daily increased from 2010, and reached the maximum (333) on 4/9/2012. None of the domains resolved to any active IP after 10/15/2013.

domains can also be nonexistent domains that never resolve. Therefore, we cannot rely solely on infrastructure to derive the set of domains D_A. Although NXDOMAINs will not be used to compute financial loss in Section 3.6.2, we would like to understand the evolution of the botnet's infrastructure. Our intuition is that, if a NXDOMAIN is queried by a large percentage of known infected hosts, it is likely to be an ad-abuse domain. However, in order to be more confident about this, we do not label it as ad-abuse domain unless there is at least one other unknown NXDOMAIN sharing the same group of infected hosts.

We use an aggressive filtering process to find such domains based on internal *host overlaps*. The internal *host overlap* is the percentage of the infected hosts that query any of the domain names in a cluster. Multiple infected hosts may query the same NXDOMAIN for reasons other than concurrent TDSS/TDL4 infections. To avoid misleading overlaps, we examined ad-abuse domains in our ground truth to derive a cutoff for strongest overlap signal. Figure 3.7d depicts the cumulative distribution of host overlaps for all ground truth ad-abuse domains. It demonstrates a plateau in the top 5% of ad-abuse domains, which we effectively use as a threshold to filter clusters and add new domains to the set D_A. After sorting the clusters with NXDOMAINs based on the host overlap, we keep the top 5% clusters. We then apply the same technique to sort all the domains in these clusters and keep the top 5% *of all domains*. This aggressive cutoff only keeps NXDOMAINs with the *strongest* host overlaps, which is in line with our lower-bound goal.

Correctness of Spectral Expansion Module

We bootstrapped the spectral expansion process with 296 TDSS/TDL4 domains gathered from various public resources. After operating Algorithm 1 2,590 times, going over every day of the NXDOMAIN dataset twice, we discovered 838 new TDSS/TDL4 domains. This means that the total number of TDSS/TDL4 domain names in the set D_A was 1,134. Next, the sanitization process reduced D_A to 765 domains based on historical WHOIS (WHOWAS) information from DomainTools. These domains match known TDSS/TDL4 domain registration email addresses or name servers, as shown in Table 3.2. The reader should note that the lookup volume for these domains will be used for the financial analysis in Section 3.6.2.

Table 3.2: Categories of newly detected ad-abuse domains. There are only three non TDSS/TDL4 domains based on manual analysis. The email addresses are obfuscated.

	Detected	**Labeled**	**Lookup Vol. (millions)**
Share Email Address			
email1@nhjhajsukk.cc	216	12	425
email2@aol.com	73	63	205
email3@dikloren.biz	65	9	144
email4@rocketmail.com	112	9	64
email5@kraniccky.com	6	3	57
email6@u7.eu	0	171	261
email7@gmx.com	0	20	28
Share TDSS Name Server	6	-	4
No Active IP Address			
Sinkholed	64	9	
Two TDSS Parking Services	25	-	
Never Registered	268	-	
Non TDSS/TDL4	3	-	
Total	838	296	

We manually analyzed the rest of the domains and found that only three domains were mistakenly added to the set D_A by the spectral expansion module, while the rest were related to ad-abuse. The category "No Active IP Address" in Table 3.2 contains domains that only resolved to known sinkholes, parking IPs, and domains that were never registered.

"Sinkholed" represents domains sinkholed by researchers. "Two TDSS Parking Services" refers to domains registered later in the four year time period, and pointed to the same two parking services used by known TDSS domains during the same time. Lastly, 268 of newly detected domains were never registered. However, based on the large host overlap of these domains with known TDSS domains and name string characteristics, we concluded that these domains were related to the TDSS/TDL4 botnet.

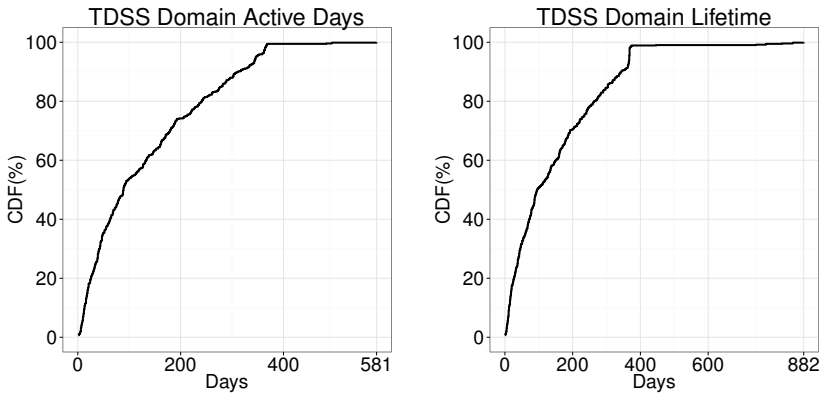
Overall, the spectral expansion Algorithm 1 was able to produce a high quality set of TDSS/TDL4 domains (D_A) while introducing a low number of non-TDSS/TDL4 domains (3 out of 838 new domains). After removing these three domains, we used the remaining 1,131 to analyze ad-abuse C&C infrastructure. The 765 domains that survived the sanitization process were used to analyze the financial impact to the ad ecosystem.

3.6 Ad-abuse Reports

This section discusses the two reports that summarize the network infrastructure properties behind the ad-abuse component of TDSS/TDL4 and our estimation around financial impact that the botnet brought to the advertisers over four years.

3.6.1 C&C Infrastructure

Using the 1,131 domains in set D_A, we analyzed the network infrastructure used by the ad-abuse component of the botnet. We separated IP addresses used by these domains into parking, sinkhole, and active categories. Besides well-known parking and sinkholing IPs, we consider IPs with more than 1,000 historical domains to be parking IPs because of the α threshold discussed in Section 3.5.2. All other IP addresses were considered to be *active*. Figure 3.8 shows the number of domains resolving into each category over the four year observation period. In total, *at least* 863 domains were registered and the botnet used 228 IP addresses. These IP addresses were used for two years and ten months, until 10/15/2013. These domains were mostly active before the middle of 2012. We should note that during



(a) Number of active days.

(b) Number of days between last and first active day.

Figure 3.9: Ad-abuse C&C domains lifetime.

July 2012, a number of researchers started sinkholing some of the TDSS/TDL4 domains. This perhaps forced the botmasters to change monetization tactics as security researchers were investigating the ad-abuse component.

Half of the ad-abuse C&C domains resolved to different *active* IP addresses for more than 90 days in total, as we can see from Figure 3.9a. Moreover, 20% of the C&C domains were active for more than 240 days. The ad-abuse domains often switched status between NXDOMAIN, pointing to active IPs, and being parked. The number of days between the last and first active day of C&C domains is the period of time that they could be successfully monetized, during which the botmasters should have received a sizable amount of traffic volume from the victims. Figure 3.9b shows the cumulative distribution of time during which the ad-abuse domains were monetized. The plot shows that 30% of domains were monetized for more than six months, and 7% of domains were monetized for more than a year.

The botnet used a variety of hosting infrastructures to facilitate the abuse. We obtained ASN information [71] for 195 out of 228 total active IP addresses used by the ad-abuse C&C. They are under 49 different Autonomous System Numbers (ASN), 59 CIDRs and 24 countries. Table 3.3 shows the distribution of the servers around the globe, used by

Table 3.3: The top 7 countries where C&C infrastructure has been identified. They count towards 71% of the IP addresses.

Country	IP Addresses	%
RU	42	18.42
US	34	14.91
LV	20	8.77
PT	19	8.33
DE	18	7.90
EU	17	7.46
NL	13	5.70
Other (17)	32	14.04
Unknown	33	14.47
Total	228	100.00

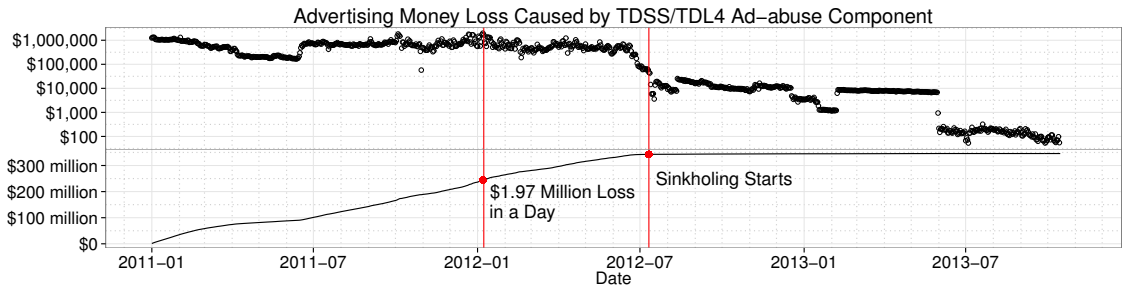


Figure 3.10: Top: Daily advertisers' money loss caused by the ad-abuse component of TDSS/TDL4. Bottom: Cumulative financial loss for advertisers. Less than 15% of the botnet population is estimated to have been involved in ad fraud that cost at least $346 million from 1/1/2011 to 10/15/2013.

TDSS/TDL4 domains.

3.6.2 Financial Analysis

We used Equation (3.2) to estimate the advertisers' financial loss. For our local network (the US ISP) we calculated the DNS Ad-abuse Rate to be $\zeta = 27.62$ (Section 3.5.1) and the percentage for impression fraud as $p_{im} = 99.13\%$ (Section 3.4.1). We used $CPM = \$2$ according to a survey over twelve ad networks [72]. We calculated the daily number of DNS requests R_i to domains used for ad-abuse that resolved to *active* IP addresses. We should note that this is an *under-estimation* since we used aggressive thresholds to exclude potentially parked domains in our passive DNS traces (as we discussed in Section 3.5.2).

This resulted in 1.2 billion DNS requests in total. μ_{im} denotes the number of ads returned by each Protocol 1 request (which relates to impression fraud activity). During our analysis, we identified instances where as many as 50 ads were returned from the C&C after each Protocol 1 request. We never saw fewer than 5 ads per request according to network traces of malware execution reported by [66]. Therefore, we used $\mu_{im} = 5$ for our lower bound estimate.

Throughout the lifetime of TDSS/TDL4, we estimate levels of ad-abuse on the order of at least $346 million using Equation (3.2). This lower bound is only based on the DNS datasets from the ISP network we had access to. Figure 3.10 shows the distribution of the financial loss caused by TDSS/TDL4 to advertisers. The daily financial loss is shown at the top of the figure, and the cumulative financial loss is at the bottom. We observed 1,018 days of active ad-abuse C&C DNS communications, caused by victims in the US ISP. This resulted in an average of $340 thousand daily financial loss for advertisers. However, before the first sinkholed domain was registered on 7/11/2012, the daily estimate was on average $616 thousand and peaked to $1.97 million on 1/7/2012. After the sinkholing action, the financial impact to the advertisers drastically decreased as the plateau of the bottom plot in Figure 3.10 shows.

We strongly believe that other networks in the world were affected by this threat based on our sinkhole analysis described in Section 3.5.1. The victims in the *entire* ISP roughly accounted for 30% of the total botnet population in the US. The infected hosts in the US were less than 50% of the entire botnet population in the world. Thus, our lower bounds may only conservatively estimate loss caused by less than 15% of the entire botnet population.

Cost For Operating The TDSS/TDL4 Infrastructure: The ad-abuse hosting infrastructure was located in 228 different IPs. Without knowing the hosting plans actually used by the botmasters, we have to consider an average cost plan for each service provider to approximate the cost of running the TDSS/TDL4 botnet. Using manual analysis, we con-

Table 3.4: Financial break down approximation among the entities of the online ad ecosystem, in millions of dollars.

Stakeholders		Money (millions)
Advertisers' Capital		346.00
DSP	45%	155.70
Ad Exchange (inbound)	8%	27.68
Ad Exchange (outbound)	8%	27.68
Ad Networks	32%	110.72
Ad Server/Publisher (Affiliates)	**7%**	**24.22**

clude that the average minimum (i.e., the botmaster is using the least expensive plan) cost is approximately $33.62 per month, whereas the average maximum cost is almost $444 per month. We assume infrastructure is used around the clock. For IPs that we could not link to a particular AS, we assume a flat rate. This rate corresponds to the median of the observed prices around the world. Using this information, we conclude the cost to operate the TDSS/TDL4 C&C infrastructure to be between $44,000 and $260,000 over four years.

Potential Financial Reward for the Botnet Operators/ Affiliates: While is impossible to know for sure what the exact levels of their reward may have been, we will try to approximate the revenue that went to the affiliate TDSS/TDL4 entities. To derive the stakeholder and the break-down described in Table 3.4 we consulted a CTO of a large Demand Service Platform (DSP) company, who wishes to remain anonymous. According to his expert opinion, these are the most typical breakdowns to various entities in the ad ecosystem. As we can see from Table 3.4, the potential financial reward for the affiliates is in the order of tens of millions of dollars.

It is logical to assume that the botmasters and affiliates are most likely getting paid as publishers or traffic resellers. In this role, the estimated revenue is 7% of money spent by advertisers, $24.22 million. The TDSS/TDL4 botnet may have a team of operators that have previously harvested this immense amount of money from the ad-abuse operation. Our estimates are in-line with investigations from law enforcement on the amount stolen by fraudulent advertisement campaigns [5, 6]. For example, law enforcement agencies

Table 3.5: The extent to which TDSS/TDL4 has affected the Internet. The tables are limited to the top 6 observations. Ad networks and Publishers domain names have been aggregated to the owner companies.

(a) Countries Affected

Country	**Infections**	**%**
US	33,386	46.77
EU	7,479	10.48
DE	5,939	8.32
CA	5,399	7.57
FR	3,369	4.72
UK	2,355	3.30
Other (168)	15,802	22.13
Total	71,374	100.00

(b) Ad Networks Targeted

Ad network	**Hits** ($\times 10^3$)	**%**
Google	1,141	14.27
Facebook	1,108	13.86
Microsoft	275	3.44
YouTube	238	2.97
Yahoo	186	2.33
PubMatic	163	2.04
Other (83,854)	4,883	61.05
Total	7,997	100.00

(c) Publisher Websites Affected

Publisher	**Hits** ($\times 10^3$)	**%**
YouTube	451	5.64
Yahoo	365	4.56
Google	337	4.20
Facebook	244	3.05
Microsoft	207	2.58
eBay	195	2.44
Other (112,889)	6,197	77.57
Total	7,997	100.00

recently estimated a minimum level of financial gains on the order of $14 million for the botmasters behind the DNSChanger botnet [73]. Note that DNSChanger was a significantly smaller botnet that operated over less than half the time period that TDSS/TDL4 was active.

Ad Networks and Publishers Targeted

Although we did not use fraudulent clicks in our financial estimations, we use sinkhole traffic from Protocol 2 to deteremine if the botnet was targeting a single or multiple entities in the ad ecosystem. We studied the distribution of ad networks whose ads have been replaced, according to the Protocol 2 traffic (Section 3.4.1). According to Table 3.5b, in total, 83,860 different ad networks were targeted. Google Inc. ad networks account for almost 17.3% (1.37 million instances) of the total ad-abuse observations. Facebook comes

second, at 13.8% (1.1 million instances). Other ad networks share the remaining 68.9%. The mean number of replaces ads was 95 and the median was four, reflecting a wide variety of replacement behavior. Even excluding the most popular ad networks targeted, there is still a long tail of 4,883 different ad networks constituting 61.05% of our observations. This shows that botmasters did not target only one ad network but rather a variety of them.

Finally, Table 3.5c shows the number of times an ad shown on a publisher's website was replaced. The top publishers had approximately 2% to 5% of all malicious ad replacements. In total, 112,895 different publisher websites were affected by the victim's malware. Google again ranks high with an overall replacement frequency of almost 10%, while their rival, Yahoo, was affected less than half as often. There is also a long tail of 112,889 different publishers.

3.7 Discussion

Our study aims to increase the situational awareness behind botnets that employ sophisticated techniques to abuse the online ad ecosystem and hopefully motivate further research in the space of ad-abuse. In this section we will discuss the most important challenges we faced while analyzing TDSS/TDL4.

3.7.1 Ground Truth Behind The Financial Loss

The botnets that interact with and monetize the ad ecosystem typically do not target a single entity (i.e., Google, Facebook, or Microsoft etc.). Due to the secrecy within the ecosystem, it is very hard to gather all the datasets from different entities necessary to verify whether the abuse levels we estimated are actually what the advertisers lost. For example, however unlikely it may be, we cannot exclude the possibility that some percentage of the impression fraud could have been detected and stopped by some entities in the ad ecosystem. Unfortunately, we cannot determine how much impression fraud, if any, was blocked, nor by whom. Thus, we had to rely on our own assumptions to estimate the lower bound. How-

ever, even in the scenario where one entity had perfect defenses, we cannot reliably assume it to be true for all the other entities in the ad ecosystem. For instance, if one DSP lacks proper defenses, fraud will still occur in the ad ecosystem. This means, that advertisers using this DSP will pay a hefty price due to ad-abuse. While we contacted several entities in the ad ecosystem, they remain secretive about the methodology and tools that they use to detect fraud. Even if a small percentage (e.g., 30%) of the reported fraudulent traffic evades detection, the losses are still significant.

3.7.2 Ground Truth Behind TDSS/TDL4

Our goal was to get ground truth around the way the TDSS/TDL4 botnet operates in the wild without contributing to online abuse. To that extent, we decided to gather the ground truth from external reports, and also from analyzing the sinkholing datasets of DGA domain names that supported the monetization module in TDSS/TDL4. Observation of DNS Ad-abuse Rate was made passively from actual infected hosts around the world. The TDSS/TDL4 victims were notified by a community effort behind this sinkholing operation, the sinkhole data were released to the operational community and several entities in the online ad ecosystem from the moment the sinkhole operation began.

3.7.3 Smart Pricing Data For Impressions and Clicks

We used Equation (3.2) to compute the lower bound of financial loss of advertisers, which assumes that perfect smart pricing for CPC was successfully used across the ad ecosystem, and all fraudulent impressions impacted the advertisers.

We chose $CPC = \$0$ to reflect on the lower bound for the financial analysis assuming CPC smart pricing was perfect. The attackers most likely can still profit from fraudulent clicks after smart pricing. For instance, recent work shows the actual CPC charged after smart pricing was between 10 to 30 cents for ZeroAccess [7]. Smart pricing is hard since not all conversion rates can be effectively measured. Not all conversion actions were logged

and shared between advertisers and ad networks/exchanges. The fact that TDSS/TDL4 does both impression and click fraud implies that the monetization technique tried to avoid detection by generating positive click-through rates.

We chose to account for all the impressions since impression fraud is still a hard problem to date. The new standard of Ad Viewability has been announced and deployed to prevent advertisers from spending money on invalid ad impressions [4, 3]. However, since there is almost no documentation about how impression fraud was handled by ad networks and ad exchanges when TDSS/TDL4 was active (before October 2013), it is reasonable to assume that a significant portion (if not all) of the impressions most likely went undetected. As we have already pointed out, getting ground truth around this would require a collaboration among many entities in the ecosystem. Such task was not realistically achievable by the authors of this study.

3.8 Related Work

TDSS/TDL is a widely spread and intensively monetized malware family that has grown, upgraded, and evolved into one of the most sophisticated rootkits over the years [74, 75, 76, 67]. The latest version (TDL4) varies significantly from its ancestor (TDL3), mainly because it is no longer limited to 32-bit systems, but can also infect 64-bit systems. The extent to which the malware managed to propagate gave its operators the opportunity to move to businesses other than ad-abuse, such as leasing the botnet and providing an installation channel for new malware to the already infected systems.

Operating a sinkhole is a safe, passive way to collect data regarding network connections between malware and the servers they try to contact. Sinkholing works for network connections that go over DNS. Malware needs to find a way to contact its Command and Control (C&C) server [77], which cannot always be done through Peer to Peer (P2P) protocols, since network administrators often block them. Therefore, DNS is the preferred channel for cheap communication. In the case of TDSS/TDL4, the malware uses P2P as

an alternative communication method [78]. Data collected from a sinkhole operation can be used to measure the network behavior of a botnet. For example, [69] used a similar approach as the one described here to uniquely identify infected hosts.

There have been several studies of click fraud abuse, including many that measure the abuse and propose counter measures. Such works often focus on the ad network's perspective [79, 68]. Springborn et al. [65] studied pay-per-view networks and described how millions of dollars are lost by fraudulent impressions annually. This loss has also been studied by Daswani et al. [80] through the "Clickbot.A" botnet of 100,000 hosts, showing how the value chain of ad-abuse operates online. Moreover, Stone-Gross et al. [54], studied abuse from both a botnet's and ad network's point of view, showing the large amount of money the botnet can make. These works carefully focus on specific parts of the ad ecosystem, while ours characterizes overall abuse impact by using edge-based metrics.

The work most similar to ours is the recent ZeroAccess study [7] that estimated daily advertising losses caused by the botnet by analyzing just one week of click fraud activities during a takedown against the ad-abuse component of ZeroAccess. This was the first study that analyzed the levels of ad-abuse behind ZeroAccess, mainly from the view of a single ad network. While the ZeroAccess study was novel, it did not help large network administrators independently measure the levels of ad-abuse originating from their network environments and take appropriate actions. Studying the ad-abuse phenomenon and deriving generalizable results cannot be achieved by using short temporal windows, or examining the problem from the viewpoint of a single ad network. Our system addresses these limitations from previous studies by studying the ad-abuse problem passively at the edge of the Internet over a multi-year time period.

3.9 Summary

We present a novel Ad-abuse Analysis System (A^2S) to conservatively estimate the long-term damage the monetization component of botnets can cause to advertisers. We studied

one of the most notorious botnets that monetized the ad ecosystem: TDSS/TDL4. Using a long-term study of four years, taken from an edge-based system that permits generalization to the overall ad ecosystem, we revealed the properties and evolution of the botnet's infrastructure. We also estimated the lower bound for the abuse: less than 15% of the botnet population inflicted financial loss to the advertisers of at least $346 million, observed via a US ISP network over four years. This reveals the extent of the abuse that botnets bring to the advertisers in the long term, making them the low risk and high reward monetization method for modern botmasters. The estimated lower bound suggests the importance of more research effort in the problem to detect and prevent it.

CHAPTER 4

MEASURING NETWORK REPUTATION IN THE AD-BIDDING PROCESS

4.1 Motivation

In Chapter 3, we propose a new clustering technique to efficiently measure the impression fraud from the botnet TDSS/TDL4. Similar to other past research efforts, the focus is detecting ad abuse at the edge (i.e., the infected host), or, "outside" of the ad ecosystem. This is vantage point V_1 in Figure 2.1, Section 2.2. However, little is known about the network policies that are being enforced *within* the ad ecosystem, especially during the ad bidding process. Advertisers do not want to display ads on low quality publishers that may include automated visits from adware and affiliate marketing entities, and thus they need to selectively respond to ad bidding requests based on the reputation of the publishers. Unfortunately, little work has been done to measure reputation of publisher domains.

In this Chapter, we examine if open source intelligence data from the security community can be used to ascertain publisher reputation. To this end, we analyze anonymized ad bidding requests between a large demand side platform (DSP) in North America and six ad exchanges over a period of three months (vantage point V_2 in Figure 2.1, Section 2.2). Using open source intelligence from public blacklists and malware execution traces, we investigate the reputation properties of publishers in the *advertisement bidding process* (Section 4.5). This Chapter makes the following key observations:

- We explain the ad bidding process and measure it in detail to improve the network and security communities' understanding of the advertising ecosystem. These measurements include bidding request traffic from six large ad exchanges for request volume, publisher domains, and client distribution. We find that malicious publisher domains tend to be present on more ad exchanges and reach more clients than non-blacklisted

publisher domains on average. These differences are statistically significant and suggest that reputation systems for advertisement publishers are possible.

- We identify that of all publisher domains seen in the DSP, 13,324 (0.27%) are on blacklists, which generate only 1.8% of bid requests, and 134,262 (2.74%) are queried by malware. This underestimates the amount of ad abuse based on other studies [23, 24], which has been measured as high as 30%. This also indicates that traditional sources of maliciousness used in the security community are insufficient to understand ad abuse seen from DSPs.

- Using graph analysis, we demonstrate how to track advertising infrastructure over time. To focus on potentially malicious campaigns, we use a simple suspiciousness heuristic based on open-source intelligence feeds. Using this technique, we identify case studies that show ad network domains support Potentially Unwanted Programs (PUP), rely on domain name generation algorithms, and are occasionally used to distribute malware.

4.2 Real-Time Bidding

In this section, we briefly describe the real-time bidding process. Figure 4.1 shows a simplified view of the Real-Time Bidding (RTB) process. The JavaScript from the publisher page requests an ad through a *bid request*. In a request, the publisher includes information such as category of the page, size of the ad space, country, user's browser and OS version, cookie, etc., and sends it to the ad exchange (step 1).

Once the ad exchange receives the bid request from a seller, it then consolidates the request into seller site information (e.g., URL of the publisher page), device information, and user data. The ad exchange sends the bid request to its buyer applications (step 2), for instance, through a DSP.

After receiving the bid request, the buyer replies with a bid response containing the ad

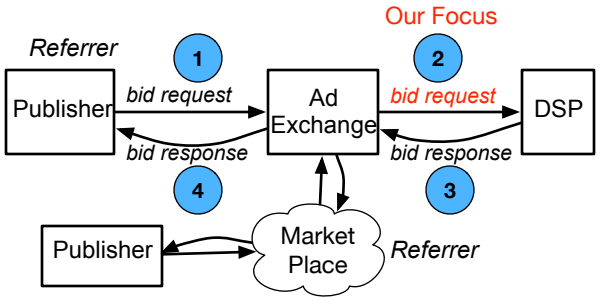


Figure 4.1: A simplified view of the Real-Time Bidding process.

Table 4.1: Summary of all datasets.

	Date Range	Size
DSP Traffic	12/10/2014 - 3/24/2015	2.61T
Public Blacklists	12/9/2009 - 1/15/2016	22G
Malware	1/1/2011 - 11/17/2015	136G
Alexa	12/10/2013 - 3/24/2015	10G
DNS	12/10/2014 - 3/24/2015	1.54T

URL and the markup price (step 3). The RTB protocol typically waits for a fixed amount of time (e.g., 100ms) to collect bids, and then chooses the winning bid under the auction's rules (e.g., OpenRTB [81]). The ad exchange then notifies the winner and returns the ad to the publisher (step 4).

In the aforementioned example, the bid request comes from the publisher directly. Therefore, the publisher page is the *referrer* for the bid request. Very often, the bid request comes from the market place, where the original request was purchased and resold by many intermediaries. In that case, the *referrer* is the last entity that sold the ad inventory to the ad exchange. Ad exchanges do not have visibility of the user-side publisher if the request comes from the market place. This is one of the challenges for ad exchanges to detect and stop fraud.

4.3 Datasets

In this section, we describe the datasets we obtained including Demand Side Platform provider (DSP) traffic, public blacklist data, and malware domain data. Table 4.1 provides

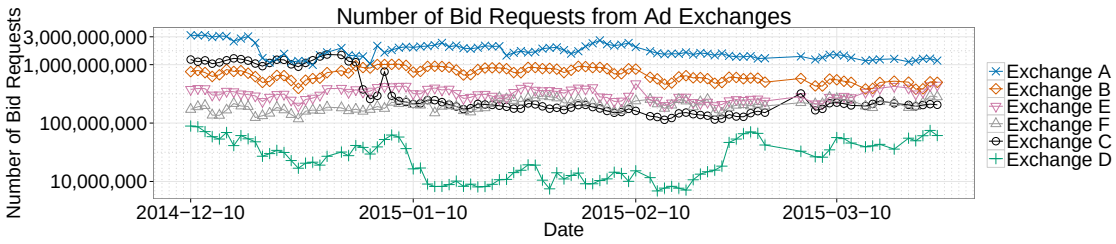


Figure 4.2: Number of daily bid requests from ad exchanges seen in the DSP.

a brief summary of the datasets.

4.3.1 DSP Traffic

The DSP provides ad bidding logs extracted from step 3 of Figure 4.1. The traffic is aggregated into eight fields per hour every day: the **ad exchange** that issued the bid request, the **publisher domain name** of the referrer URL, the **hashed IP address** of the user, the **country code** and **autonomous system number** of the IP address, the hourly **timestamp** of when the bid request was sent, and lastly the **number of bid requests** seen within the specific hour that match all the previous fields. Within the fields, the **publisher domain name** represents either the webpage that users saw, or the last traffic reseller before the bid request reached the ad exchange. Next, we describe DSP traffic using the volume of bid requests and publisher domain names.

Bid Request Volume

It is reasonable to assume that for each bid request, some advertiser wins the bid eventually. Therefore, the bid request volume can be considered to be the number of ad inventories purchased and shuffled through the ad exchanges from the visibility of the DSP.

Figure 4.2 shows the bid request volume from six different ad exchanges from 12/10/2014 to 3/24/2015. One of these ad exchanges is ranked top five in market share. On average, there are 3.45 billion bid requests daily in total. Individually, Exchange A processed the most bid requests of all, with an average of 1.77 billion requests per day. Exchange B

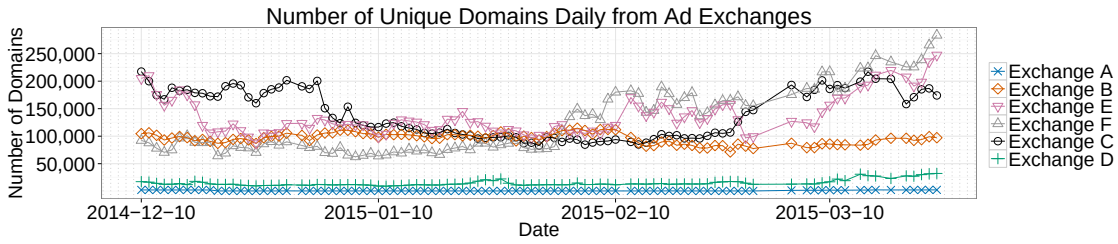


Figure 4.3: Number of daily publisher domains from ad exchanges seen in the DSP.

comes next, with an average of 695 million requests per day. In addition, Exchange E, Exchange F, and Exchange C received bid requests on the order of hundreds of millions. Finally, Exchange D had an average of 30 million bid requests daily, which fluctuated the most compared to other ad exchanges.

Comparing the volume of the last day from the DSP traffic (3/24/2015) with that of the first day (12/10/2014), there is a decline in the overall bid request volume from Exchange A (63.2%), Exchange B (34.3%), Exchange C (83.2%), and Exchange D (31.2%). However, the volume increased for Exchange E (18.34%) and Exchange F (64.26%). Our DSP confirmed that this was not a traffic collection problem but could not identify the root cause of these changes.

Publisher Domains

The publisher domain field in the DSP traffic indicates the source of an ad request. It is either the publisher website where the ad will be shown, or the reseller domain redirected from some previous publisher.

An average of 391,430 total publisher domains were seen from all ad exchanges every day. Figure 4.3 shows the number of unique publisher domains from each ad exchange. Although Exchange A had the highest number of bid requests (Figure 4.2), it represented the lowest number of unique domains (average: 955) per day. It is likely that many of them are traffic resellers. For instance, `coxdigitalsolutions.com` is a subsidiary of Cox specializing in buying and selling digital media. It is the most popular publisher domain

in Exchange A, generating more than 20% of all bid requests. The small set of publisher domains of Exchange A is quite stable. There were no new publishers in 39 days out of three months, and an average of 91 new publisher domains on the other days. Exchange D has the fewest bid requests and also had very few publisher domains, an average of 14,732 every day. If an ad exchange works with few publishers, it is easier to provision them and block malicious traffic. On the other hand, it is harder to know the source of ad inventories from reseller publishers, meaning detection may need to happen at the reseller's perspective.

Two ad exchanges saw the largest number of new publisher domains. Exchange E had an average of 22,647 new publisher domains, while Exchange F had an average of 23,405 new publisher domains daily. Towards the end of March 2015 in Figure 4.3, there were as many as 35,794 new domains from Exchange E and 56,151 new domains from Exchange F. Both ad exchanges also increased the volume of bid requests during the same time period in Figure 4.2. The churn rates of the publisher domain names in these two ad exchanges were quite high. This presents a challenge for ad exchanges to track the reputation of new publishers.

Lastly, Exchange B had a stable number of publisher domains every day, on the order of 100,000. There was a decrease in the number of daily publisher domains seen from Exchange C around the end of 2014, and then the number increased again, reaching the 150,000 mark towards the end of March 2015.

4.3.2 Other Datasets

In order to measure reputation in the DSP bid request traffic, we also obtained other datasets that provide threat information, which includes public blacklists and dynamic malware execution traffic. Both provide insight into known abuse in the ad exchanges. We crawled seven public blacklists [82, 83, 84, 85, 86, 87, 88] daily from 12/9/2009 to 1/15/2016. In total, 1.92 million unique domains appeared on the public blacklists. Dynamic malware

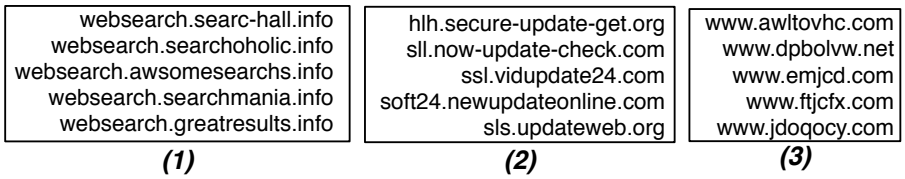


Figure 4.4: Examples of blacklisted publisher domains seen in the DSP traffic.

execution feeds are from one university [89] and two industry partners. The binaries were each executed for five minutes in a controlled environment. We extracted date, malware md5, and the domain names queried during the execution of the binaries. The feeds are collected from 1/1/2011 to 11/17/2015. There are 77.29 million unique malware md5s, querying a total of 14.3 million domain names. We use *PBL* to denote the public blacklists dataset and *Md5* to denote the malware domains dataset.

Lastly, we collected DNS resolution data every day from a passive DNS repository in North America between 12/10/2014 to 3/24/2015. The dataset contains domain name, query type, and resolved data every day for A, NS, CNAME, and AAAA query types. We observed a daily average of 891 million unique mappings between domain names. On average, the DNS resolution dataset matches 71.56% of all publisher domain names seen in the DSP in the same day. Among the 28.55% publisher domains from DSP not seen in passive DNS, the majority of them are long tail content sites. For example, unpopular blog sites, user's own fantasy sport pages, customized lists pages, etc. Long tail content can be specific to certain users' interests and not commonly accessed across different networks. In full disclosure, this is perhaps the only not fully open source intelligence source we used in our experiments. However, commercial passive DNS offerings are very simple to obtain today [90]. We will use the resolution information to construct infrastructure graphs and track them over time in Section 4.6.

4.4 Fraudulent Publisher Domains

In this section we provide examples of blacklisted publisher domains that generated ad bidding requests through the ad exchanges. These domains are from adware and affiliate marketing programs.

4.4.1 Case 1: PUP

Blacklisted publisher domains can be generated by Potentially Unwanted Programs (PUP) such as browser hijacker and pop-up ads.

Figure 4.4 (1) shows domain names of pattern `websearch.*.info` that are used by browser hijackers [91]. The adware forces the user to use a different search engine to steal impressions that would have otherwise been delivered through typical search engines (e.g., Google, Bing, Yahoo, etc.). The adware hijacks user search queries and makes ad bidding requests from these publisher domains to generate revenue.

Figure 4.4 (2) shows "update" domains used by pop-up ads. The adware shows pop-up ads that masquerade as fake updaters for legitimate software, such as Windows, Flash, and video players [92]. These publisher domains make ad bidding requests from pop-up windows generated by the adware.

4.4.2 Case 2: Affiliate Marketing

Blacklisted publisher domains may represent affiliate marketing domains. These affiliate domains request ads through ad exchanges on behalf of adware or malware. We manually analyzed network traces from dynamic execution of malware md5s that contained domains in Figure 4.4 (3). The malware uses fake referrers to send HTTP GET requests through domains in Figure 4.4 (3). Then the requests go through a chain of redirections until finally receiving an ad to generate revenue.

4.5 Measurement

We first discuss client IP location distribution in DSP traffic in Section 4.5.2. Then, we perform reputation analysis of publisher domains by correlating them with blacklists and malware domains in Section 4.5.3.

4.5.1 Summary of Findings

In summary, we found that:

- There are 13,324 (0.27%) known malicious domains generating bid request traffic through the ad exchanges in our datasets. On average, they generate 1.8% of overall bid requests daily, much less than previously published values [23, 24]. However, 68.28% of blacklisted domains were identified by public blacklists before they appeared in DSP traffic. This suggests traditional sources of maliciousness are valuable, but insufficient to understand ad-abuse from the perspective of DSPs.

- On average, blacklisted publisher domains tend to use more ad exchanges (average: 1.85) and reach more clients (average: 5109.47) compared to non-blacklisted domains (average ad exchanges: 1.43, average hashed client IP addresses: 568.78) (Section 4.5.3). This suggests reputation systems for ad publishers are possible.

- Contrary to the observation of blacklisted publisher domains, malware domains use a similar number of ad exchanges (average: 1.44), but are seen from more hashed client IP addresses (average: 2310.75), compared to publisher domains never queried by malware (average ad exchanges: 1.43, average hashed client IP addresses: 485.36). (Section 4.5.3)

4.5.2 Client Analysis

We observed 436 million hashed client IPs that sent bid requests for ads. According to information provided by the DSP, the hashed client IP addresses are from 37,865 different

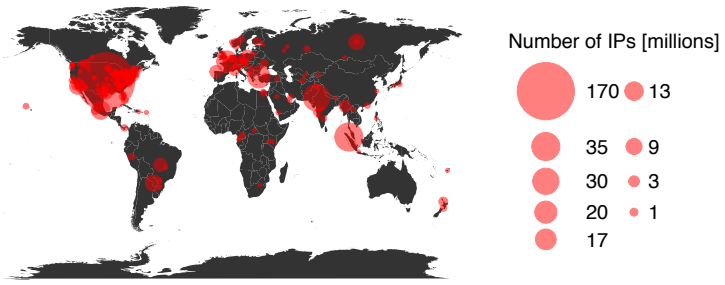


Figure 4.5: Distributions of client IP address locations.

Table 4.2: 4.2a: The top six countries for 66.75% of hashed client IP addresses. 4.2b: The top six Autonomous System Names for 17.66% of hashed client IP addresses.

(a) Client Location

Country	**Hashed IPs** millions
US	174 (39.91%)
GB	35 (8.03%)
DE	31 (7.11%)
CA	21 (4.82%)
FR	17 (3.90%)
MX	13 (2.98%)
Other	103 (23.62%)
Unknown	42 (9.63%)
Total	436 (100.00%)

(b) AS Name

AS Names	**Hashed IPs** millions
Comcast	18 (4.13%)
AT&T	17 (3.90%)
Deutsche Telekom	14 (3.21%)
MCI	12 (2.75%)
Verizon	9 (2.06%)
Uninet	7 (1.61%)
Other	359 (82.34%)
Unknown	42 (9.63%)
Total	436 (100.00%)

Autonomous Systems in 234 different countries.

Table 4.2a shows the top six countries where hashed client IP addresses reside. Nearly 40% of clients are located in the United States. Next, it is the United Kingdom with 8% of hashed IP addresses. The top six countries also include Germany (7.11%), Canada (4.82%), France (3.90%), and Mexico (2.98%). There is a long tail of 228 other countries for the remaining clients. Overall the top six countries account for 66.75% of all the hashed client IP addresses seen in DSP. Figure 4.5 shows the country distribution of hashed client IP address locations.

Table 4.2b presents the top six Autonomous System Names (ASNs) for hashed client IP addresses. The ASN distribution is less biased compared to the country distribution. Comcast, AT&T, and Deutsche Telekom are the top three ASNs, each with under 5% of all

hashed IP addresses. There are 37,859 different ASNs in the long tail of the distribution, which contains 82.34% of all hashed IPs.

4.5.3 Reputation Analysis

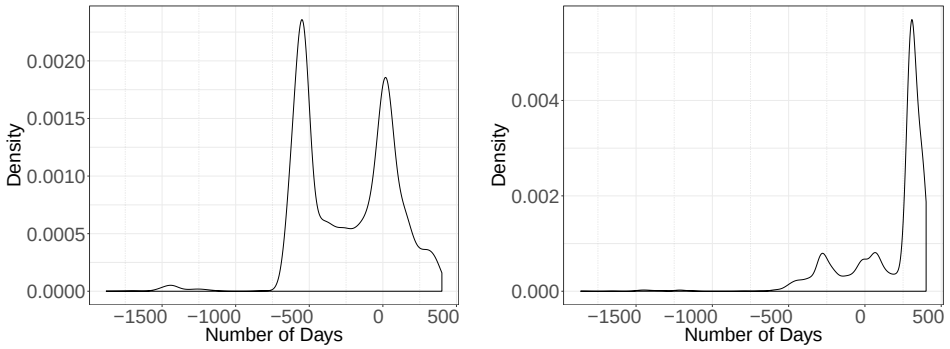
In this section, we explain how we intersect publisher domains from DSP traffic with blacklists and malware domains to perform reputation analysis.

Public Blacklist Traffic

Since 89.87% of the domains on the blacklists we collected do not have semantic information, we filter them to ensure they are bad publishers with high confidence. We want to be conservative about what we keep, so we choose the following filters. First, we obtained all the domains that appeared on the Alexa [93] top one million list for every day from 12/10/2014 to 3/24/2015. We excluded those consistent Alexa domains because they are unlikely to be malicious. Second, we excluded all domains under the ad server category of EasyList [94], because malware conducting impression fraud or click fraud can generate traffic that goes through ad servers. Lastly, we excluded a hand curated a whitelist of CDN effective second level domains (e2lds) and we excluded all fully qualified domain names that overlapped with these e2lds.

Observation 1: **0.27% publisher domains appeared in DSP traffic were blacklisted by the security community. They generated 1.8% of all bid requests daily.**

We observed 4,905,224 unique domains in the DSP traffic from 12/10/2014 to 3/24/2014. Among them, 13,324 (0.27%) domains were blacklisted some time between 12/9/2009 and 1/15/2016. Blacklisted domains were responsible for an average of 1.8% of all bid requests every day. Previous studies estimate nearly 30% of bid requests are malicious [23, 24], which suggests this is only a fraction of the actual abuse. While there are many potential causes, such as referrer spoofing or lack of ad-abuse investigations, these findings show simply relying on blacklists from the security community is insufficient to study and com-



(a) First seen dates (b) Last seen dates

Figure 4.6: Density plot of first seen date date on PBL - first date seen from DSP (4.6a) and last seen date on PBL - last date seen from DSP (4.6b).

bat abuse. While they are few, we investigated the potential to automatically detect these abusive domains.

***Observation 2:* 68.28% of blacklisted publisher domains were known to the security community before they appeared in DSP traffic.**

Figure 4.6a shows the density distribution for the difference of days between when a domain was first blacklisted and when it was seen in DSP traffic. The zero value in this case means that the domain name was blacklisted on the same day as it was seen in the ad exchanges. Similarly, a value of -500 means that the domain was blacklisted 500 days before it ever appeared in the datasets from the DSP. The plot shows that 68.28% (9,097) of all blacklist domains were known to the security community prior to they started requesting for ads in the DSP traffic. Moreover, 32.49% (4,329) of blacklisted publisher domains were labeled more than 535 days before they were seen in the DSP datasets. The peaks of the distribution reflects several blacklists update events. One event was a major update of 4,031 domains on 6/23/2013, which corresponds to the -535 days in Figure 4.6a. Another update event on 12/4/2014 was reflected around -6 days in the plot. Eighty domains were blacklisted on 1/15/2011, which makes up the small bump around -1500 days in the plot.

Figure 4.7 is a scatter plot of the first date a domain is blacklisted (x-axis) and its corresponding first seen date in the DSP (y-axis). The size of the point represents the

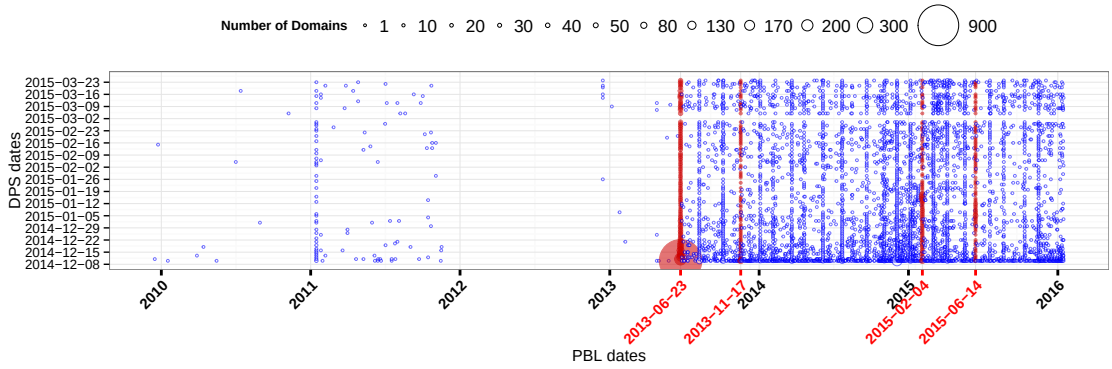


Figure 4.7: Scatter plot of first date seen on PBL and first date seen from DSP for all DSP domains that were on PBL.

number of domains in these dates. The points in the bottom side of the plot are large because this is the first date we had the DSP data. The vertical group of points represent domains being updated in the blacklist in the same day. We highlighted a few days when blacklisted domains from the DSP traffic were first labeled. The plot is more dense on the right side since 2013-06-23. We increased the number of blacklists to crawl from 3 to 7 on that day, which resulted in more domain names in PBL dataset and more overlap with the DSP traffic from that point on. On 2013-11-17, the blacklists updated many domain names including `websearch.*.info` used by browser hijackers. On 2015-02-04, there were a lot of "update" domains used by pop-up ads added to the blacklists, e.g., `soft12.onlineupdatenow.com`. On 2015-06-14, the blacklists updated a group of algorithmically generated domains with sub domains `freempr#`.

Observation 3: **Most (77.01%) blacklisted publisher domains remained on blacklists after they were last seen in DSP traffic.**

We would like to see whether the publisher domains remained on the blacklists after they were seen in the DSP. We plotted the density distribution for the number of days when a domain was last seen on blacklists minus when it last appeared in the DSP (Figure 4.6b). The distribution has shifted a lot towards the right part of the x-axis this time. Figure 4.6b shows that the majority (77.01%) of blacklisted domains were still on blacklists after they

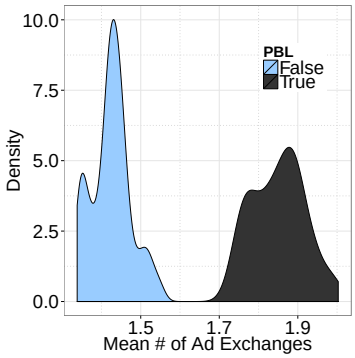
were seen in the DSP. A total of 14.06% (1,873) of them remained on blacklists more than a year after they were last seen in the DSP datasets. The peak of Figure 4.6b reflects the last date (1/15/2016) of our blacklist dataset. Overall 8,051 DSP domains belong to this peak in the plot.

Observation 4: **Blacklisted publisher domains tend to use more ad exchanges and reach more hashed client IP addresses than those that have never been blacklisted.**

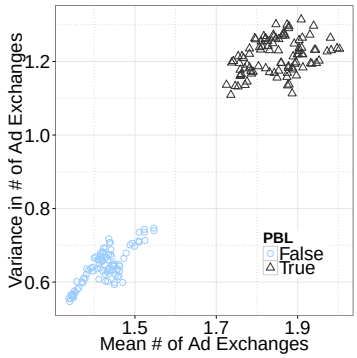
Each day, we separate the publisher domains into two groups: those that were seen in PBL (True) and not in PBL (False). For each group, we compute the average number of distinct ad exchanges and the number of hashed client IPs that a publisher domain was seen from, as well as the variance within the group. We visualize the results in Figure 4.8a to Figure 4.8d.

Figure 4.8a shows the density distributions of the daily average number of ad exchanges for the PBL group and non-PBL group across the entire DSP dataset. The PBL group were seen from an average of 1.7 to 2 ad exchanges, more than the non-PBL group. We perform a two-sample Kolmogorov-Smirnov test (K-S test) where the null hypothesis is that x=y, i.e., that the datasets are drawn from the same distribution. The K-S test demonstrates we can reject this null hypothesis ($p - value < 2.22 * 10^{-16}$). Therefore, the two distributions are significantly different. We also plot the mean and variance of the average ad exchange number for each group in Figure 4.8b. The figure shows that not only do non-PBL domains use fewer ad exchanges in general, the difference of the measure between non-PBL domains is small, as reflected by the variance. On the other hand, PBL domains have relatively higher variance among themselves.

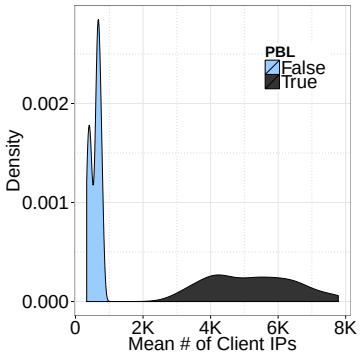
Similarly, we plot the density distribution for number of average hashed client IP addresses in a day for the PBL and non-PBL groups (Figure 4.8c), as well as the mean and variance of the metric (Figure 4.8d). These figures show that PBL domains tend to be seen from more hashed client IPs than non-PBL domains. Since the majority of the content on the web is in the unpopular "long tail", only a few hashed client IPs visit any non-PBL



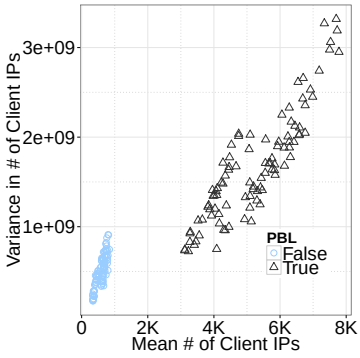
(a) Pbl: # of Ad Exchanges Density



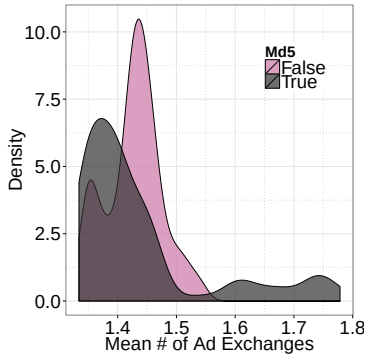
(b) Pbl: Mean, Variance for # of Ad Exchanges



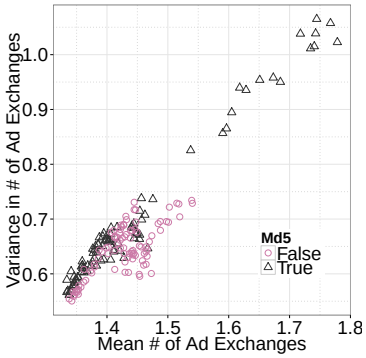
(c) Pbl: # of hashed IPs Density



(d) Pbl: Mean, Var for # of hashed IPs

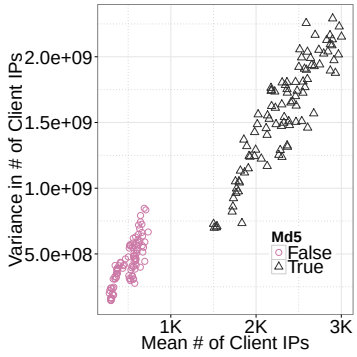


(e) Md5: # of Ad Exchanges Density

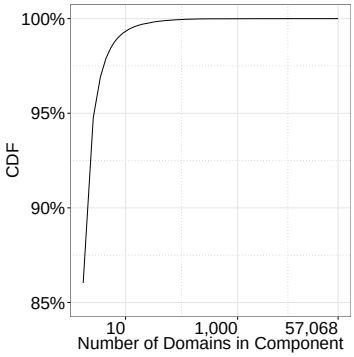


(f) Md5: Mean, Variance for # of Ad Exchanges

(g) Md5: # of hashed IPs Density

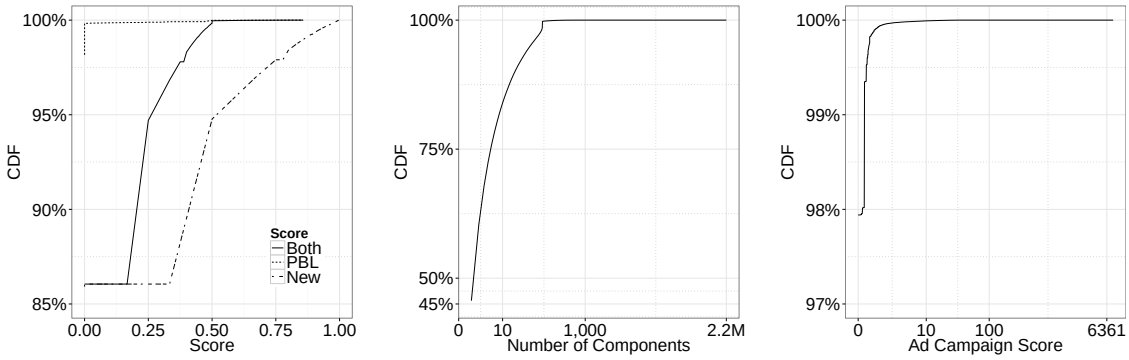


(h) Md5: Mean, Var for # of hashed IPs



(i) Components of 12/10/2014

Figure 4.8: Figure 4.8a to Figure 4.8d are PBL plots. Figure 4.8e to Figure 4.8h are Md5 plots. Figure 4.8i shows the CDF for number of publisher domains forming components of 12/10/2014.



(a) Three scores on 12/10/2014

(b) Component sizes in ad campaigns

(c) Ad campaign scores

Figure 4.9: Figure 4.8i to Figure 4.9c are three scores for components seen on 12/10/2014 (Figure 4.9a), number of components in ad campaigns (Figure 4.9b) and ad campaign scores (Figure 4.9c).

domain in general, and the variance of number of clients is low (Figure 4.8d). In contrast, PBL domains seen in the RTB process aim to make money, and thus spread to as many hosts as possible.

Malware Traffic

Domains queried by malware are another type of threat information commonly used by the security community. We filtered the malware domains using the same three methods as in the PBL case. Within 4,905,224 unique domains from the DSP traffic, 134,262(2.74%) were queried by malware samples collected over five years. *There are ten times more publisher domains queried by malware than from those on blacklists.* Similarly, we can separate the publisher domains into two groups: malware domain group (Md5 True) and non-malware domain group (Md5 False). We computed the average daily number of ad exchanges and hashed client IP addresses for each day in the DSP traffic.

Observation 5: **Malware domains have different behavior than blacklisted domains. That is, malware domains were observed to employ similar number of ad exchanges to non-malware domains, however, with a higher number of hashed client IP addresses.**

Figure 4.8e to Figure 4.8h show the measurement results. We observe bimodal distributions of malware vs. non-malware domains in Figure 4.8e and Figure 4.8g. Figure 4.8e and Figure 4.8f show that publisher domains queried by malware tend to use a similar number of ad exchanges. In addition, the distributions between malware domains and non-malware domains overlapped much more than when we compared PBL group with non-PBL group. Therefore, the number of ad exchanges is not a distinguishing attribute for the MD5 group. On the other hand, DSP domains queried by malware were still seen from a larger group of hashed client IP addresses, compared to the rest of domains never queried by malware. Malware domains that interact with ad ecosystem are relatively more popular than non-malware domains.

Malware query non-malicious domains for various reasons, and only a few of the domains are fraudulent publishers. Recall that when malware interacts with the ad ecosystem from the client side (Figure 2.1), there may be syndicated publishers, or benign ad servers contacted by the malware, in order to reach ad exchanges. Despite our filtering efforts, it is likely that there are still numerous benign domains in the malware domain set. Additionally, domains could remain on blacklists after they become inactive or parked, which results in false positives when using blacklists. These findings all point to the need for better ad-abuse ground truth datasets.

4.6 Infrastructure Tracking

In this section, we show that traditional DNS infrastructure features can be used to extend the ground truth set, discover new ad abuse cases and track the threat evolution over time. This can be used by any entity in the ad ecosystem with visibility of bidding requests to track advertising campaign infrastructure—focusing on those that are likely to be malicious in intent. While we acknowledge that the word "campaign" has an overloaded meaning, we define it in the following way and only in the context of ad abuse: *a campaign will be defined as the set of domain names that can be linked together over time based on their IP*

infrastructure properties.

At a high level, we construct graphs of the relationship between the domain name of the ad publisher and the infrastructure the domain name uses. By building and merging these graphs over time, we can track the infrastructure and focus on those campaigns that may be malicious, e.g., domains known to have been blacklisted, queried by malware, or have never been seen before. We present case studies based on this process in Section 4.7.

4.6.1 Constructing Infrastructure Graphs

An *infrastructure graph* is an undirected graph G, defined by its set of vertices V and edges E. A *disconnected* graph is made up of multiple *components* or subgraphs with no adjacent edges between them. These components correspond to advertising campaigns that are tracked over time. Vertices in infrastructure graphs are domain names or the RDATA the domain names resolve to. RDATA can be an IPv4/IPv6 address (A/AAAA), a canonical name (CNAME), or a nameserver (NS). Two vertices are adjacent if and only if exactly one is a domain name, and the domain name resolved to the RDATA of one of the aforementioned query types (A/AAAA/CNAME/NS) during time t when the domain name appeared as a publisher for a bid request.

A Demand Side Platform provider (DSP) can build infrastructure graphs by performing the following steps. First, the DSP collects all publisher domain names D_p from the bid requests seen on day t. Second, the DSP resolves all domain names $d \in D_p$, which results in zero or more domain name and IP address tuples. More formally, resolving d will yield $[(d, rdata_0), \cdots, (d, rdata_N)]$ if d resolves to N different IPs, CNAMEs, or NSes on day t. Each of these tuples corresponds to an edge in our graph G. Finally, after G is built for day t, G is decomposed into its *connected components* C, where each component $c \in C$ is ranked and tracked over time as a specific ad campaign. While we experimented with more sophisticated community discovery or spectral methods, the benefits gained were disproportional to the add-on complexity. Thus, we decided to select the simplest and most

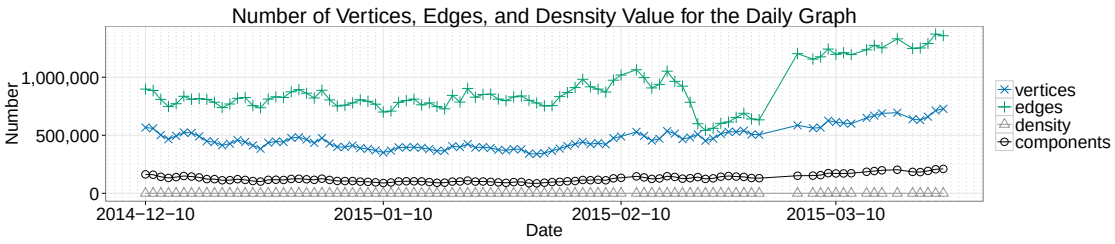


Figure 4.10: Number of vertices, edges and density values for the graph every day.

straightforward way to mine the graph for campaigns.

Since the DSP bidding request traffic did not include DNS resolution information, we chose to correlate that with the DNS dataset obtained from a passive DNS database from a North American ISP (Table 4.1). By combining the DNS resolution seen in the same day in the ISP with the publisher domains from the bidding request traffic, we were able to construct daily infrastructure graphs. Next, we discuss how we analyze the produced graphs.

Graph Analysis

We study the infrastructure graphs using some basic graph analysis metrics. Specifically, we first analyze overall graph properties including vertices, edges and density measures. Then, we examine the connected components of the graphs every day and over time. These analytics help us understand the infrastructure of the publisher domains, and give us insights about how to rank components based on how suspicious they are and track them over time.

First, we discuss three properties of daily infrastructure graphs. Figure 4.10 shows three statistics for graphs generated every day: number of vertices (V), number of edges (E), and the density measure. We use the following formula to compute the *graph density D*:

$$D = \frac{2E}{V(V-1)} \tag{4.1}$$

On average, there are 472 thousand vertices, and 883 thousand edges every day. The

graphs are extremely sparse and the daily density is only $8.35 * 10^{-6}$. In fact, the majority of the edges only connect two vertices. There are 566,744 vertices on 12/10/2014, and it dropped to 342,426 (by 39.58%) on 1/29/2015. Then the number of vertices slowly increased to 727,501 on 3/24/2015. Since vertices include publisher domains and DNS resolution data, the change in the number of vertices over time is largely consistent with the observation of how the number of daily publisher domains changed (Figure 4.3). On the other hand, the change in the number of edges per day is different. The number of daily edges decreased since 2/17/2015, and dropped to the lowest number 542,945 on 2/21/2015, before it jumped up to 1,203,202 on 3/5/2015. Through manual analysis, we concluded that this was not caused by any single domain name. There were fewer resolved data per domain in general in these days.

Second, we study properties of connected components in the infrastructure graphs. Figure 4.10 shows the number of connected components over time that were in the daily infrastructure graphs. On average, there are 127,513 connected components in a day. Figure 4.8i demonstrates that the daily infrastructure graph is highly disconnected. The cumulative distribution for the size of the components in a day follows the Zipf's law. For instance, CDF in 12/10/2010 shows that 86% of connected components have only one publisher domain in it. Fewer than 0.7% components have more than ten publisher domains.

4.6.2 Identifying Suspicious Components

The number of graph components based on the results from Section 4.6.1 can be hundreds of thousands in a day (Figure 4.10), which is likely too many for manual analysis. However, the measurement from Section 4.5 suggests we can prioritize components that are likely to be interesting from a security perspective. We know publisher domain names differ in behavior when they are known to appear on blacklists. Conversely the subset of malware domains seen in DSP are very noisy, and thus it is not a good metric to use for prioritizing components. We also hypothesize that never-before-seen domains deserve close scrutiny

as they may represent infrastructure changing to avoid detection. The question remains if these are indicative of true malicious behavior. To find out, we rank publisher components by their domain names, specifically, if they are on blacklists, if the domains have never been seen before and a combination of these two measures.

For each publisher component $c \in C$ we compute two values β_c and ν_c that correspond to the proportion of domains in c that appear on blacklists, and are under brand new from the perspective of the DSP, respectively. Intuitively, the first one indicates an association with known malicious activity, and the last suggests the potential threat may have just begun. Specifically, the way we compute each value of a component is smoothed.

$$\beta_c = \frac{\#\ of\ blacklisted\ publisher\ domains - 1}{Total\ \#\ of\ publisher\ domains} \tag{4.2}$$

$$\nu_c = \frac{\#\ of\ brand\ new\ publisher\ domains - 1}{Total\ \#\ of\ publisher\ domains} \tag{4.3}$$

We offset the numerator count by one based on results of the infrastructure graph analysis from Section 4.6.1. Since the majority of components have only one publisher domain name in it, they are isolated singletons and do not provide any information to other unlabeled domains from infrastructure point of view. We prefer not to prioritize these singletons among all components even if they are already blacklisted or brand new. Equation 4.2 and Equation 4.3 give singleton components both zero values. Moreover, we judge whether a domain name is "brand new" using the effective second-level domains (e2ld) according to public suffix list [95]. An e2ld is the smallest registrable unit of a domain name and two domains under an e2ld are likely operated by the same individual. Therefore, a new domain under a new e2ld is more interesting to us.

After getting these two values β_c and ν_c, we also compute the linear combination of these: $\iota_c = \frac{1}{2}(\beta_c + \nu_c)$. Finally, we reversely sort the components in a day based on the ι_c score. Within a day, ι_c can range between 0 and 1. A component with higher ι_c

will be prioritized over a component with lower ι_c for inspection. Figure 4.9a presents cumulative distributions of the proportion of pbl-related, never-before-seen domains and a linear combination of the two for a day per component. A total of 98% of the components have zero PBL score because they do not have any blacklisted domains, and 14% of the components have a score for having new domains. The final component score combining the two falls in between the two distributions.

4.6.3 Tracking Campaigns Over Time

Building infrastructure graphs for an individual day is useful, but tracking the ad campaigns over time will yield more comprehensive coverage of ad campaigns, as well as advanced warning of potentially malicious ones. First, if an ad campaign is determined to be malicious, tracking them over time through small infrastructure changes will enable more comprehensive blacklists to be built. Second, if a tracked ad campaign is known to be malicious, newly added infrastructure can be more pro-actively blacklisted. Finally, tracking infrastructure over time allows us to build ground truth to eventually model malicious and benign advertising campaign infrastructure. In our future work we plan to experiment with predicting fraudulent publishers.

To unify ad campaigns across multiple infrastructure graphs, we simply join ad campaigns that share IP addresses, canonical names, and name servers that are the same. This allows us to not only construct graphs within days, but also across time. We will show that this simple tracking method works well in practice. While on average there are 127K connected components every day, only 10K of them form new ad campaigns. A DSP can choose to only go through top-ranked new components if there is limited time available for threat analysts.

ι_{ad} is used to sort advertising campaigns to identify case studies. It is calculated by adding up all the interesting scores of individual components ι_c belonging to that campaign. After we sort the ad campaigns by ι_c, we then examine the distribution of the

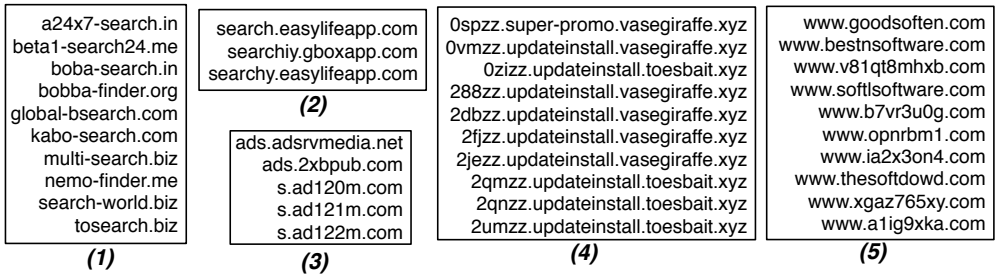


Figure 4.11: Publisher domain examples.

interesting scores and number of components in the campaigns. Figure 4.9c shows the cumulative distribution of ad campaign scores. Also, Figure 4.9b shows the CDF of the number of components in an ad campaign. Overall 99.99% ad campaigns have fewer than 1,000 components. The ad campaign with the largest number of components (2.2 million in Figure 4.9b) has the highest ad campaign score. Domains in this campaign resolved to several parking, and sinkholing IP addresses, as well as common names servers like GoDaddy. This is the reason that this noisy campaign is not representative of maliciousness or freshness of the domains. Starting from the second ad campaign, the interesting score indicate suspicious activities in the ad exchanges. We now describe the case studies this measure uncovers in Section 4.7.

4.7 Case Studies

Among the campaigns with highest (top 0.1%) interesting scores, we found new cases including Potentially Unwanted Programs (PUP), algorithm generated domains and malware sites.

4.7.1 Case 1: PUP

Among advertising campaigns with the highest interesting scores, one category of publisher domains are generated by Potentially Unwanted Programs (PUP). For example, domains in Figure 4.11 (1) to (5).

A VirusTotal report [96] suggests a machine communicating with domain names in Fig-

ure 4.11 (1) (ι_{ad} ranked the 3rd highest) is likely infected with a trojan known as LEMIR or Win32.BKClient by the AV industry. The malware has many capabilities including changing default search engines to generate revenue, disabling Windows AV, Firewall and Security Center notifications, and can drop additional malicious binaries [97]. Similarly, ad campaigns with 2^{nd} and 4^{th} highest ι_{ad} (Figure 4.11 (2) (3)) are generated by ad injections of certain browser extension. Different malware families communicate with domains in Figure 4.11 (3) including Win.Trojan.Symmi [98]. These publisher domains may not be malicious, but they are strongly associated with monetization behavior of malware. These are interesting cases as traditional malware are involved in an area where we would expect to see only adware or "potentially unwanted programs." This shows that malware uses advertising fraud to monetize infections and malware can also be identified from the vantage point of a DSP.

In addition, several Pop-up Ads campaigns exhibit high level of agility similar to traditional malware. The ad campaign ranked $1,184^{th}$ (Figure 4.11 (4)) uses domain fluxing, likely to avoid browser extension detection systems. In total, we observed more than 26,000 unique domain names from this campaign in three months of DSP traffic. Moreover, the ad campaign in Figure 4.11 (5) not only uses domain fluxing, it also uses the Amazon EC2 cloud to further decrease the chance of detection. Each of these domains resolved into an EC2 cloud domain representing a unique Virtual Machine (VM), when active. The VM domains also change according to the domains that point to them. This shows that miscreants are constantly employing fresh VMs to perform ad fraud. Since traditional detection systems often use reputation of IP addresses of domains and URLs, using cloud machines makes this campaign harder to be detected.

4.7.2 Case 2: Algorithm Generated Domains

Figure 4.12 (1) (2) shows two ad campaigns of algorithm generated domains we found in the DSP traffic (ranked 142^{th} and 183^{th}), containing at least 195 domains. None of the

Figure 4.12: Malware site example.

domains were blacklisted, but a high percentage of brand new domains results in a high score. A new group of domains appear everyday, pointing to the same IP address. These publisher domains are suspicious. Although no open threat analysis evidence is available to date, it is reasonable to assume that anything that changes so often must be trying to evade a detection process. With infrastructure tracking, ad exchanges or DSP can keep a close eye on such campaigns to proactively deal with potential ad abuse.

4.7.3 Case 3: Malware Site

Figure 4.12 (3) shows a group of malware site domains (ranked $1,484^{th}$ campaign) seen from DSP traffic, none of which appeared on blacklists. A Virustotal report [99] shows that the IP address these domains resolved to, had other similar domains pointing to it during the week ending on 3/24/2015. Related URLs were detected as malware sites by several URL scanners from the AV industry. This group uses domain fluxing with both the second level domain zone, and the child labels. We saw other groups of domains tracked separately, with similar domain name patterns, and short lifetime. However, they were not grouped into one big campaign, because different groups were using different IP addresses. In other words, this campaign uses both domain fluxing and IP address fluxing. Since we only used exact the same IP address match to form a campaign, we will need other information to further analyze campaigns like this.

4.8 Related Work

Previous research has studied behavior of click bots [80, 100, 68]. The bots mimic human behavior by generating fake search queries and adding jitters to click delay. More advanced bots hijacked users' original clicks and replaced the ads [68, 62, 7, 101]. The ZeroAccess botnet cost advertisers $\$100,000$ per day [7] and the TDSS/TDL4 botnet cost advertisers at least $\$346$ million in total. Ad fraud detection work mainly focused on click fraud [102, 103, 104].

Impression fraud is harder to detect than click fraud. Springborn et al. [65] studied pay-per-view networks that generated fraudulent impressions from invisible iFrames and caused advertisers millions of dollars lost. Advertisers can purchase *bluff ads* to measure ad abuse [68] and compare charged impressions with valid impressions. The adware and ad injection problem has been systematically studied by static and dynamic analysis of web browser extensions [105, 106, 107]. From within the ad ecosystem, Stone-Gross et al. [54] used ad hoc methods to study specific attacks faced by ad exchanges, including referrer spoofing and cookie replay attacks. Google also documented what they consider to be invalid traffic in [47] but did not disclose the details of their traffic filters.

4.9 Summary

In this Chapter, we measured ad abuse from the perspective of a Demand Side Platform (DSP). We found that traditional sources of low reputation, such as public blacklists and malware traces, greatly underestimate ad-abuse, which highlight the need to build lists catered towards ad-abuse. The good news, however, is malicious publishers that participate in ad-abuse can likely be modeled at the DSP level based on their behavioral characteristics. Finally, malicious campaigns can be tracked using graph analysis and simple heuristics, allowing DSPs to track suspicious infrastructure.

CHAPTER 5

PRACTICAL ATTACKS AGAINST GRAPH-BASED CLUSTERING

5.1 Motivation

Several studies have shown how security systems that employ machine learning techniques can be attacked [8, 9, 11, 12, 13], decreasing their overall detection accuracy. This reduction in accuracy makes it possible for adversaries to evade detection, rendering defense systems obsolete.

While graph based network detection systems are not immune to adversarial attack, the community knows little about *practical attacks* that can be mounted against them. As these network detectors face a range of adversaries (e.g., from script kiddies to nation states), it is important to understand the adversary's capabilities, resources, and knowledge, as well as the cost they incur when evading the systems.

In this Chapter we present the first practical attempt to attack graph based modeling techniques in the context of network security. Our goal is to devise generic attacks on graphs and demonstrate their effectiveness against a real-world system, called Pleiades [25]. Pleiades is a network detection system that groups and models unsuccessful DNS resolutions from malware that employ domain name generation algorithms (DGAs) for their command and control (C&C) communications. The system is split into two phases. First, an unsupervised process detects new DGA families by clustering a graph of hosts and the domains they query. Second, each newly detected cluster is classified based on the properties of the generated domains.

To evade graph clustering approaches like Pleiades, we devise two novel attacks—targeted noise injection and small community—against three commonly used graph clustering or embedding techniques: i) community discovery, ii) singular value decomposition

(SVD), and iii) node2vec. Using three different real world datasets (a US telecommunication dataset, a US university dataset and a threat feed) and after considering three classes of adversaries (adversaries with minimal, moderate and perfect knowledge) we mount these two new attacks against the graph modeling component of Pleiades. We show that even an adversary with minimal knowledge, i.e., knowing only what is available in open source intelligence feeds and on their infected hosts, can evade detection.

Beyond devising practical attacks, we demonstrate that the attacks are inexpensive for adversaries. Fortunately, defenders are not without recourse, and detection systems' parameters can be tuned to be more resistant to evasion. Based on these discoveries, we make recommendations to improve Pleaides' resilience.

This Chapter makes the following contributions:

Two Novel Attacks The *targeted noise injection attack* improves on prior work that randomly injects noise; by targeting the injected vertices and edges to copy the graph structure of the original signal, we force noise into the resulting clusters. Our *small community attack* abuses the known property of small communities in graphs to subdivide and separate clusters into one or more unrelated clusters.

Practical Attacks and Defenses While more knowledgeable attackers typically fare better, we demonstrate that even minimal knowledge attackers can be effective: attackers with no knowledge beyond their infections can render 84% of clusters too noisy to be useful, and evade clustering at a rate of 75%. The above attacks can be performed at low cost to the adversary by not appearing to be anomalous, nor losing much connectivity. Simple defenses raise the attacker's costs and force only 0.2% of clusters to be too noisy, and drop the success rate to 25%. State of the art embeddings, such as node2vec, offer more adversarial resistance than SVD, which is used in Pleiades.

5.2 Related Work

Existing work in adversarial machine learning has focused on analyzing the resilience of classifiers. Huang et al. [108] categorize attack influence as either *causative* or *exploratory*, with the former polluting the training dataset and the latter evading the deployed system by crafting *adversarial samples*. Following the terminology of Huang et al., our work focuses on *exploratory* attacks that target the graph clustering component of Pleiades. We assume that the clustering hyperparameters are selected with attack-free labels, and the subsequent classifier is not polluted when they are trained. Contrary to other *exploratory* attacks in literature, we face the challenge that the clustering features cannot be modified or computed directly, and that attackers often have an incomplete view of the defender's data.

In order to compute optimal graph partitions or vertex embeddings, one needs to have a *global* view of all objects on the graph. On the contrary, related work can compute classification features directly from crafting adversarial samples. For example, features are directly obtained from spam emails [8, 9], PDF files [10, 11, 12], phishing pages [11], images [14, 15, 13, 16], network attack packets [17], and exploits [18, 109]. These security applications classify an object based on features extracted from only that object and its behavior. This makes the features of system classifiers more *local*, and enables evasion techniques such as gradient descent directly in the feature space. We make the following definition: a *local* feature can be computed from only one object; whereas a *global* feature needs information from all objects being clustered or classified.

Since Pleiades uses *global* features, an adversary's knowledge can affect the success of attacks. For example, if the adversary has full access to the defender's datasets, she can reliably compute clustering features and is more equipped to evade than a less knowledgeable attacker. Many researchers [110, 111] have shown that, even without access to the training dataset, having knowledge about the features and an oracle to obtain some labels

of objects is sufficient for an attacker to approximate the original classifier.

Biggio et al. [19, 20] are the first to study adversarial clustering. They propose a bridge attack, which works by injecting a small number of adversarial samples to merge clusters. The attackers have perfect knowledge in their assumption. We distinguish our work by i) considering attackers with different knowledge levels, ii) evaluating how adversarial graph-clustering in network security affects the whole system, and iii) quantifying the cost of attacks. With respect to attack cost analysis, Lowd et al. [112] propose a linear cost function as a weighted sum of feature value differences for crafting evasive adversarial samples. Since we do not work directly in the feature space, we propose different costs for the attacks we present in Section 5.3.

To summarize, our work is novel because we focus on adversarial clustering, which deals with global features that cannot be directly changed. We also evaluate capabilities of attackers with various knowledge levels, and quantify the costs associated with attacks.

5.3 Threat Model & Attacks

In this section, we describe our threat model and explain our attacks as modifications to a graph G. In practice, the attacker changes the graph based on the underlying data that are being clustered. For example, if the vertices in a graph are infected hosts and the domains they query as in Pleiades, the graph representation can be altered by customized malware that changes its regular querying behavior.

5.3.1 Notation

An undirected graph G is defined by its sets of vertices (or nodes) V and edges E, where $G = (V, E)$ and $E = \{(v_i, v_j) : \text{if there exists an edge between } v_i \text{ and } v_j, v_i \in V, v_j \in V\}$. An *undirected bipartite graph* is a special case where V can be divided into two disjoint sets (U and V) such that every edge connects at a vertex in U and one in V, represented as $G = (U, V, E)$. While the attacks apply in the general case, oftentimes bipartite graphs appear in

security contexts: hosts (U) query domains (V), clients connect to servers, malware make system calls, etc. Finally, a *complete* undirected bipartite graph is where every vertex in U has an edge to every vertex in V.

$\mathcal{G}$ is an undirected graph that represents the underlying data a defender clusters. The graph clustering subdivides $\mathcal{G}$ into clusters $C_0, \ldots, C_k$, where $V = C_0 \cup \ldots \cup C_k$. If the graph clustering method is based on graph partitions, then each cluster C_i is a subgraph G_i, and $\mathcal{G} = G_0 \cup \ldots \cup G_k$. Often when applied, a defender seeks to cluster vertices either in U or V of the bipartite graph, for example, cluster end hosts based on the domains they resolve, or malware based on the system calls they make. An attacker controls an *attacker graph*, $G \subset \mathcal{G}$. The adversary uses the targeted noise injection and the small community attacks described below to change G to G', by adding or removing nodes and edges from G.

These attacks violate the underlying basic assumptions of graph clustering techniques, which either renders the clustered subgraph G' to be useless to the defender or prevents G' from being extracted from $\mathcal{G}$ intact (See Section 5.3.3).

5.3.2 Threat Model

Before describing attacker knowledge levels, we discuss knowledge that is available to all attackers. We assume all attackers have at least one active infection, or $G \subset \mathcal{G}$. The attacker is capable of using any information that could be gathered from G to aid in their attacks. We also assume that an attacker can evaluate clusters like a defender can, e.g., manual verification. When done with a classifier, an attacker has black-box access to it or can construct a surrogate that approximates the accuracy and behavior of the real classifier based on public data. This may seem extreme, but the plethora of open source intelligence (OSINT) [113, 114, 115] data and MLaaS machine learning tools [116, 117, 118, 119, 120] make this realistic. Finally, an attacker has full knowledge of the features, machine learning algorithms, and hyperparameters used in both the unsupervised and supervised

phases of the system under attack, as these are often published [25, 121, 29, 30, 28, 122, 123]. Since clustering requires some graph beyond G, we must consider attackers with various representations of the defender's $\mathcal{G}$. We evaluate three levels: minimal, moderate, and perfect knowledge. The minimal level attacker only knows what is in their attack graph G, but the perfect attacker possesses $\mathcal{G}$. For example, a perfect adversary would have access to the telecommunication network data used in Pleiades, which is only obtainable by the most sophisticated and well resourced of adversaries.

Minimal Knowledge The minimal knowledge case represents the least sophisticated adversary. In this case, only the attacker graph G is known, as well as any open source intelligence (OSINT). For example, the attacker can use OSINT to select potential data to inject as noise, or can coordinate activities between their vertices in G. In the Pleiades example, an attacker with minimal knowledge can draw information from their infected hosts.

Moderate Knowledge The moderate knowledge case represents an adversary with $\tilde{\mathcal{G}}$, an approximation of $\mathcal{G}$. If attacking Pleiades, $\tilde{\mathcal{G}}$ would be a host/domain graph from a large enterprise or university in order to approximate the view that the defender has. This allows the adversary to evaluate their attacks. The size of $\tilde{\mathcal{G}}$ affects the evaluation from the attacker's perspective, which we will explore by randomly sampling subgraphs of $\tilde{\mathcal{G}}$. An attacker with moderate knowledge is similar to a sophisticated adversary with access to large datasets through legitimate (i.e., commercial data offerings) or illegitimate (i.e., security compromises) means.

Perfect Knowledge Finally, the perfect knowledge case is for an adversary who has obtained $\mathcal{G}$ from the defender. Given the full dataset and knowledge of the modeling process, an adversary can completely reconstruct the clustering results of the defender to evaluate the effectiveness of their attacks. Ideally, this data would be well guarded making this

level of knowledge only realistic for the most sophisticated of attackers, e.g., nation-state sponsored threats. Nevertheless, considering the damage that could be done by a perfect knowledge attacker is important as a security evaluation, since it allows us to find potential weaknesses in the graph clustering techniques.

5.3.3 Attacks

We present two novel attacks against graph clustering. The first, *targeted noise injection*, improves on random injections [124, 125] by emulating the legitimate signal's graph structure. The second, *small community attack*, exploits the known phenomenon of *small communities* in graphs [42, 126]. Our attacks violate both the homophily and the structural equivalence assumptions used by graph clustering methods. That is, our attacks either change what nodes are close together to violate homophily, or they change observations of node neighborhoods so as to violate structural equivalence.

Identifying a successful attack depends on the system, which will be described in detail in Section 5.4. Since we use Pleiades, we evaluate attacks by the impact on a subsequent classification of the resulting adversarial clusters. However, this could be done purely at the unsupervised level by *manually* evaluating the accuracy of the output clusters, or leveraging prior work in adversarial malware clustering [19] to measure global cluster quality decrease. Next, we evaluate the cost incurred by the attacker. We analyze the costs by measuring changes to their graph's structure that would either flag them as anomalous or damage connectivity between the graph's vertices. In the descriptions below, an attacker's initial graph G is shown, and the alterations yield a modified graph, G', that represents a defender's view of the attacker's graph after the adversarial manipulation.

Targeted Noise Injection

Figure 5.1 illustrates two targeted noise injection attacks. Consider a bipartite attacker graph G, with vertex sets U (circles) and V (squares). To mount the attack, noise is injected

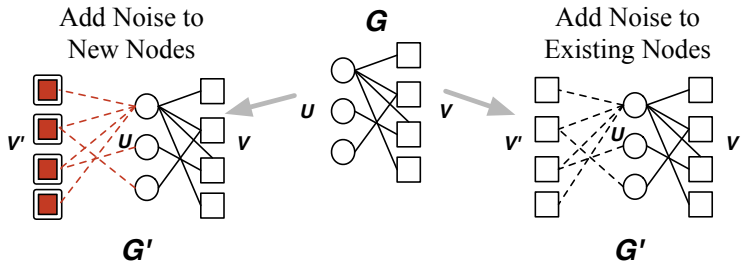


Figure 5.1: Example of targeted noise injection attacks on a graph.

into G to generate G'. We inject noisy edges from nodes controlled by the attacker for the purpose of mirroring real edges. This encourages newly connected nodes to be clustered together with the attacker's nodes.

To inject noise, the attacker creates an additional vertex set V', represented by red squares. Entities in V' should differ substantially from those in V, which depend on the underlying system to be evaded. In Pleiades' case, this means the injected domains (V') must be different, in terms of character distribution, from the legitimate domains (V). Then, for every edge between U and V, the attacker creates a corresponding edge between U and V', as shown in Figure 5.1. That is, the attack function $f : (u, v) \in E \mapsto (u, v') \in E'$ is bijective. This creates $G' = (U, V \cup V', E \cup E')$, where E' are the corresponding edges from U to V', denoted by dotted red edges in the figure. The other way to inject noise is to create edges from U to existing nodes from $\mathcal{G}$, as shown in Figure 5.1. This does not add additional nodes, but identifies other vertices on the defender's graph $\mathcal{G}$ to use as V'. A new edge is created for all edges between U and V. Attacker information is used to identify additional nodes to use. Example nodes may include other non-malicious domains queried by infected hosts, or a machine's existing behavior observed by the malware. More commonly, it requires some knowledge of the graph being clustered, $\mathcal{G}$. This process can be repeated to increase $|V'|$ to be multiples of $|V|$.

Algorithm 2 formally describes noise injection for attacker $\mathcal{A}$ controlling the attacker graph G, with noise level m. Line 1 to Line 6 repeats the noise injection process m times.

Algorithm 2 Targeted Noise Injection Attack Algorithm for Attacker $\mathcal{A}$ controlling G

Input: $\mathcal{A}, m, G = (U, V, E)$
Output: G'

1: **for** $i = 1$ **to** m **do**
2: $V_i' \leftarrow$ according to knowledge of $\mathcal{A}$
3: **for** $v' \in V_i'$ **do**
4: Mirror the edges such that $f : (u, v) \in E \mapsto (u, v') \in E_i'$ is bijective.
5: **end for**
6: **end for**
7: Return $G' = (U, (\bigcup_{i=1}^{m} V_i') \cup V, (\bigcup_{i=1}^{m} E_i') \cup E)$

In line 2, $\mathcal{A}$ generates the set of *noisy nodes* V' according to her knowledge. From line 3 to 5, the attacker creates a one-to-one mapping from E to E_i'. Line 8 returns the manipulated attacker graph $G' = (U, (\bigcup_{i=1}^{m} V_i') \cup V, (\bigcup_{i=1}^{m} E_i') \cup E)$. We will evaluate two variants to determine how much noise is needed to mount a successful, but low cost attack. In the first variant $m = 1$, and in the second $m = 2$.

While additional edges and nodes could be injected arbitrarily at random, we choose to mirror real edges in order to make both nodes from V' and V have similar embeddings. We define V' to be the set of *noisy nodes*. The targeted noise injection attack exploits the homophily assumption [38, 33] of graph clustering methods. In community discovery and spectral methods, graph partitions cannot distinguish injected noisy nodes (V') from real nodes (V), which exhibit structurally identical connections to U. The co-occurrence increases the observation of noisy nodes appearing in neighborhoods of real nodes, and vice versa for node2vec. We expect nodes from V' to join existing clusters containing V.

The targeted noise injection attack has a cost for the attacker of raising the profile of nodes belonging to attacker graph G, potentially making them outliers. Specifically, hosts from U will increase in percentile with respect to their degree, i.e., a relatively high degree could indicate anomalous behavior, which we can measure by the increase in percentile ranking changes before and after an attack. We call this the *anomaly cost*.

For attacking Pleiades, consider a graph where U are infected hosts and V are the domain names that the hosts in U query. To generate G' an attacker instructs their malware to query an additional domain ($v \in V'$) for each domain used for its normal malicious operation ($v \in V$). This causes the domains from V and V' to conflict such that the clustering is not useful to the defender. However, the anomaly cost may make these trivial to detect. Nonetheless, we will show in Section 5.5.2 that the cost of attack is small enough to be practical.

Small Community

Figure 5.2 illustrates four potential small community attacks of increasing intensity. The small community attack removes edges and/or nodes such that the graph clustering separates a single attack graph into multiple clusters, while maintaining as much connectivity from the original graph as possible. Again, G is a bipartite attacker graph with identical vertex and edge sets as before. To mount the attack, an adversary first constructs a complete version of G, $\hat{G}$. In $\hat{G}$, every vertex in U has an edge to every vertex in V. To construct G', the adversary removes edges from $\hat{G}$. In Figure 5.2, the attacker has removed one and two edges per vertex in V in G'_1 and G'_2, respectively. Then in G'_3 and G'_4, the attacker has removed a vertex from V, and then removed one and two edges per remaining vertex. The attacker randomly chooses n_v (such that $0 \leq n_v \leq |V| - 1$) nodes to remove, and n_e (such that $0 \leq n_e \leq |U| - 1$) edges from each remaining node V in $\hat{G}$. In the extreme case, there is only one vertex remaining from V connecting to one in U, which often cannot be captured by graph embeddings. Each attack instance is configured with (n_v, n_e) pair, or, in other words, the ($|V| - n_v$, $|U| - n_e$) pair to keep nodes and edges. We define the *attack success rate* as the number of successful attack configurations divided by $|U| * |V|$.

If the attacker has minimal knowledge, she can choose n_v and n_e randomly, and hope for the best. With perfect knowledge (knows $\mathcal{G}$), she can choose the smallest n_v and n_e that succeed. Attackers without some knowledge or approximation of $\mathcal{G}$ will be unable to

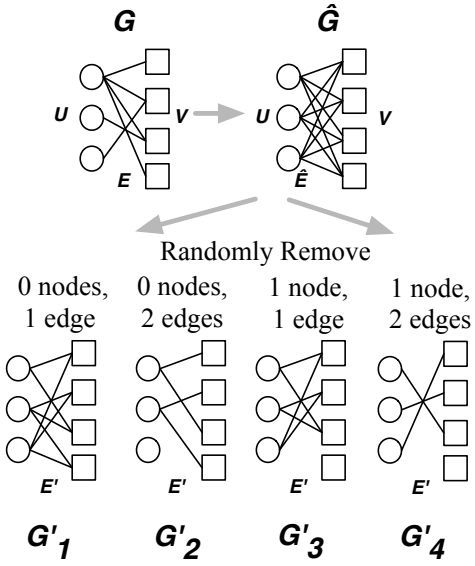


Figure 5.2: Example small community attacks on a graph.

Algorithm 3 Small Community Attack Algorithm for Attacker $\mathcal{A}$ controlling G

Input: $\mathcal{A}, G = (U, V, E)$
Output: G'

1: Construct $\hat{G} = (U, V, \hat{E})$ from G, where $|\hat{E}| = |U| * |V|$
2: $n_v, n_e \leftarrow$ according to knowledge of $\mathcal{A}$, where $n_v < |V|, n_e < |U|$
3: $V' \leftarrow$ Choose $|V| - n_v$ random nodes from V
4: **for** $v' \in V'$ **do**
5: Choose $|U| - n_e$ random edges that connect to v' to update U' and E'
6: **end for**
7: Return $G' = (U', V', E')$

verify if their attacks succeed. While G could be manipulated directly, removing nodes and edges lowers the utility for the attacker by losing connectivity in their attack graph. We aim to reduce the average edge number per node of V in G, while simultaneously maintain the lowest possible cost. Constructing and altering $\hat{G}$ both simplifies the experiments of quantifying the small community attack cost and makes the job of the attacker easier. We believe this does not negate the correctness of our experiments.

Algorithm 3 formally shows the the small community attack $\mathcal{A}$ in control of graph G. Line 1 constructs the complete graph $\hat{G}$ from G. In line 2, $\mathcal{A}$ chooses (n_v, n_e) according

to her knowledge. Then, the attacker chooses $|V| - n_v$ random nodes from V as V'. Each node in V' connects to all nodes in U. From line 4 to 6, the attacker chooses $|U| - n_e$ random edges to keep for each node in V', and thus forming U' and E'. Lastly, line 7 returns $G' = (U', V', E')$ as the manipulated attacker graph.

The small community attack exploits the information loss in graph embeddings. While community discovery works better at identifying islands and singletons, graph embeddings may miss such signal given the hyperparamters chosen at deployment. Existing methods for choosing hyperparameters do not account for potential small community attack opportunities. The downside of the attack is the *agility cost*. By removing nodes and edges from G, the adversary has to give up control over nodes, redundancy, or even functionality. In addition to losing n_v, the agility cost can be measured by the change in *graph density* (Equation 5.1) from $\mathcal{D}(G)$ to the chosen $\mathcal{D}(G')$. We define the following $D(G)$ and $D(G')$:

$$\mathcal{D}(G) = \frac{|E|}{(|U| * |V|)} \tag{5.1}$$

$$\mathcal{D}(G') = \frac{|E'|}{(|U| * |V|)} \tag{5.2}$$

A graph's density ranges from [0, 1], which denote a graph with zero edges or all possible edges, respectively. For $D(G')$ we consider how many edges are in E' compared to the maximum possible number of edges between the original U and V. This normalizes the number of edges by the structure of the G. The *agility cost* is $D(G) - D(G')$ if $D(G) > D(G')$, or zero if $D(G) \leq D(G')$. A loss in density implies a potential loss of connectivity, but maintaining or increasing the density bodes well for the attacker. They can afford an even denser structure, yet still evade defenders. It is important to note that, while n_v is lost, this is reflected in the density score, as $|V|$ includes any removed vertices like n_v. A lower density, and therefore a higher cost, is incurred when edges and/or vertices are removed relative to the original structure seen in G.

Consider an attack on Pleiades. $\hat{G}$ is created by completing G. To mount the attack like G_2', the adversary partitions the domain names that are used to control her malware by removing one of the control domains (n_v=1), and then excludes two distinct hosts that query each of the remaining domains (n_e=2). In other words, the adversary can also randomly choose one host ($|U| - n_e$) to query each one of remaining control domains. This reduces the density from $D(G) = 0.5$ to $D(G_2') = 1/3$, and sacrifices one node (n_v).

If the adversary has knowledge that allows testing whether the attack is successful or not, the attacker can increasingly remove domains and queries from hosts until clustering $\mathcal{G}$ no longer results in G' being extracted as a single cluster. In practice, as described in Section 2.1, the subdivided G' often ends up either as portions of the "death star" cluster; or in multiple, noisy clusters. In both cases, the legitimate cluster is effectively hidden in a forest of noise. In order to verify an attack was successful, however, an attacker must have $\mathcal{G}$ or an approximation.

5.4 Attacks in Practice

We chose to attack Pleiades because it has been commercially deployed and relies on graph modeling. Our reimplementation has similar performance, as shown in Appendix A.3. We now describe portions of the reimplementation in detail.

5.4.1 Pleiades

An overview of Pleiades is shown in Figure 5.3. We focus our attacks on the clustering component and use the classification phase to demonstrate attack success. First, Pleiades clusters NXDOMAINs (V) queried by local hosts (U) using the host-domain bipartite graph ($\mathcal{G}$). It groups together NXDOMAINs queried by common hosts into clusters $C_0, \ldots, C_k$, based on the assumption that hosts infected with the same DGA malware query similar groups of DGA domains. The graph clustering can be achieved by either community discovery (1), spectral clustering (2) as in the original paper [25], or node2vec clustering (3).

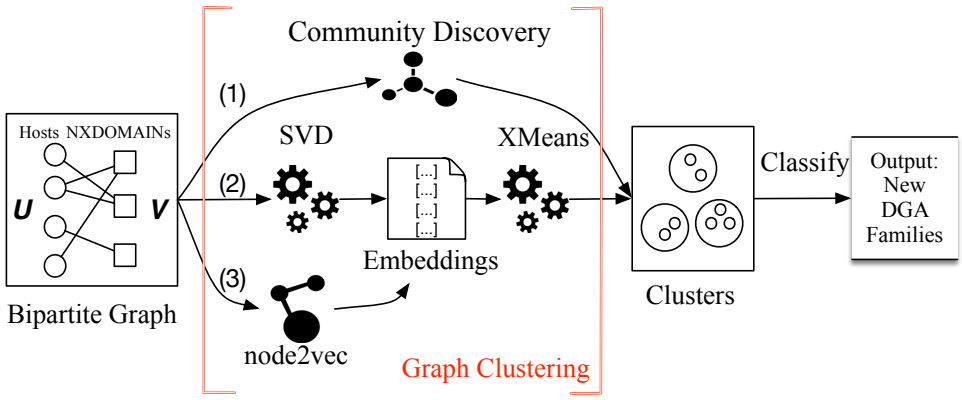


Figure 5.3: Overview of the DGA detection system.

Table 5.1: Summary of datasets and their availability to minimal, moderate, and perfect knowledge attackers.

Dataset	**Number of Records**	**Minimal**	**Moderate**	**Perfect**
Reverse Engineered DGA Domains	14 DGA Families; 395K NXD	X	X	X
Host-NXDOMAIN Graph (Surrogate)	8782 hosts; 210K NXD	-	X	X
Host-NXDOMAIN Graph (Ground Truth)	average 262K hosts; 1.8M NXD	-	-	X

Then the classification module computes domain name character distributions of each cluster into a numerical feature vector, which is used to classify known DGA families. A new unknown DGA family with features statistically similar to a known one can be detected by this process. The system operates on daily NXDOMAIN traffic generated by all hosts in a network using the following data sources.

Datasets

We use anonymized recursive DNS traffic from a large telecommunication company from December 18, 2016 to December 29, 2016. The dataset contains NXDOMAINs queried by hosts and the query timestamps. On average, there are 262 thousand unique anonymized hosts, with 44.6 million queries to 1.8 million unique NXDOMAINs in a day. We use this dataset to construct Host-NXDOMAIN Graph as ground truth without attack. This is

available to defenders and perfect knowledge attackers.

As a surrogate network dataset, we use NXDOMAIN traffic from a large US university network collected on December 25, 2016. It contains 8,782 hosts and 210 thousand unique NXDOMAINs. Among these NXDOMAINs, only 227 appeared in the ground truth network dataset. The surrogate dataset is available to attackers with moderate and perfect knowledge.

Last but not least, we use a reverse engineered DGA domains dataset to train the supervised component of the system. We run the reverse-engineered algorithms [127] to generate DGA domains for 14 malware families: Chinad, Corebot, Gozi, Locky, Murofet, Necurs, NewGOZ, PadCrypt ransomware, Pykspa, Qadars, Qakbot, Ranbyus, Sisron, and Symmi. The training dataset also includes live DGA domains observed in the ground truth network. We label 267 clusters belonging to four malware families present in the ground truth network dataset (Pykspa, Suppobox, Murofet, and Gimemo), and manually verify that these subgraphs are attack free. We train a Random Forest classifier with an average accuracy of 96.08%, and a false positive rate of 0.9%. The classifier trained from this dataset is available for attackers of all knowledge levels. Table 5.1 summarizes these datasets.

We discovered 12 new DGA malware families in only 12 days using the ground truth network traffic (see Appendix A.3 for details). We discovered real but unsuccessful evasion attempts in the wild, and retrained our classifier with evasive instances. We believe we have faithfully reimplemented Pleiades because we use comparable datasets and we achieve similar clustering and modeling results.

5.4.2 Attacks

Using the notation described in Section 5.3, let $\mathcal{G}$ be a bipartite graph of the defender. U represents hosts, both infected and uninfected, and V represent NXDOMAINs queried by hosts in the underlying network. An edge exists from $v_i \in U$ and $v_j \in V$ iff the ith host queried the jth NXDOMAIN. For an attacker graph $G \subset \mathcal{G}$, the hosts in U are infected

hosts under the control of the attacker. In the noise injection case, the attacker instructs their malware to query NXDOMAINs beyond what is needed for normal operation, as shown in Figure 5.1. In the small community case, the attacker coordinates the querying behavior of their malware such that they query fewer NXDOMAINs in common, as in Figure 5.2. We will evaluate the effectiveness of the attacks by the drop in predicted class probabilities and the predicted label of the classifier. In a Random Forest, the predicted class probabilities of a feature vector are calculated as the average predicted class probabilities of all trees in the forest. In practice, if the predicted class probability decreases substantially, the classifier will incorrectly label the instances, and the attack will be considered successful.

5.4.3 Attack Costs

To compute the anomaly cost for noise injection, we analyze percentile changes of edges related to hosts in U in the structure of $\mathcal{G}$ from before and after the attack. We quantify this change by computing the cumulative distribution functions (CDFs, example in Appendix A.1) of vertex degrees before and after a successful attack is mounted. Concretely, if an attacker can evade Pleiades but raises the profile of their infected hosts from the 50^{th} (in the CDF before attack) to the 99.9^{th} percentile of NXDOMAINs queried per host (in the CDF after attack), a defender will be able to detect such behavior with simple thresholding (i.e., monitoring hosts entering the 95^{th} percentile).

To quantify the adversarial cost behind the small community attack, we measure the change of attacker graph density $D(G')$ as defined in Section 5.5.3. If the attacker graph density decreases, this means the attacker no longer uses NXDOMAINs for their infection and/or the infected hosts query fewer NXDOMAINs in common, reducing their connectivity overall and increasing the botnet's management cost.

5.5 Results

First, we show how to select hyperparameters for each of the three graph methods. Next, we present our results for both attacks against each graph based clustering technique, for the three knowledge levels. Finally, we explain the costs incurred by the attacker, and how these can be used to identify possible defenses.

Summary of Results Our attacks against the graph clustering component of Pleiades gravely reduce the predicted class probability of the subsequent classification phase. Even with minimal knowledge, an adversary can launch an effective targeted noise injection attack dropping the median predicted class probability to 0%. In the higher knowledge levels, the maximum predicted class probability can be as low as 10%. Using a set of labeled DGA malware families observed in spectral clustering, the attacks reduce the prediction accuracy from 99.6% to 0%.

In addition to being effective, the attacks do not substantially raise the anomaly profile of infected hosts: before and after the targeted noise injection attacks the hosts occupy a similar percentile for the number of NXDOMAINs queried. Small community attack results show that the traditional way of choosing hyperparameters for generating graph embeddings is insufficient when we analyze the system in an adversarial setting, because it creates a large area for possible small community attack instances. While following the accepted methodology for selecting the rank for SVD and hyperparameters for node2vec, all DGA clusters can be hidden in noisy clusters by subdividing the infected hosts into smaller groups to sacrifice some agility, even while using hundreds of DGA domains. Even in the minimal knowledge case where the small community attack cannot be tested, attackers can sometimes still hide.

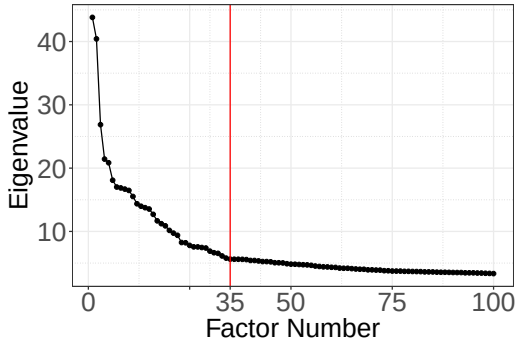


Figure 5.4: Scree plot of eigenvalues of SVD.

5.5.1 Choosing Hyperparamters

First, we carefully choose hyperparameters for the graph clustering methods in Pleiades to ensure high quality clusters are generated.

Spectral Clustering

We use the scree plot shown in Figure 5.4 to choose the rank of SVD. We fix the rank of the SVD to be 35, where the scree plot plateaus. While different than the 15 used in the original Pleiades implementation [25], the underlying datasets are different so it is not unreasonable to find different ranks.

Community Discovery

We use the best partition method from the NetworkX community discovery library [128] which implements the Louvain algorithm [33]. The Louvain algorithm efficiently extracts good communities on the graph by optimizing the modularity metric, which measures the density of links within communities in comparison to outside them. It first sets each node to be its own community, and iteratively merges them to maximize modularity. The algorithm stops when a local maxima is reached. This community detection algorithm scales to large network with hundreds of millions of nodes.

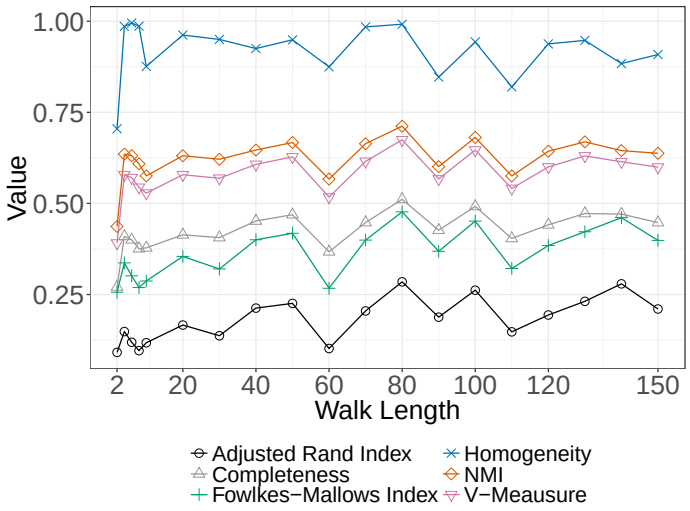


Figure 5.5: Using cluster validity metrics to choose walk length.

node2vec

We use traditional cluster validity metrics to compare different hyperparameters of node2vec. Twelve DGA malware families, including both known and newly detected ones, were used as *reference clusters*. We use validity metrics including Adjusted Rand Index, Completeness, Fowlkes-Mallows index, Homogeneity, Normalized Mutual Information (NMI), purity, and V-Measure score. We first choose context size six, which has the first highest validity scores. Several larger context sizes generate equal validity scores, but they produce noisier clusters. This is because that larger context sizes in DNS graphs tend to include more noisy nodes, such as popular benign NXDOMAINs, or popular hosts that are likely proxies.

Then we choose the walk length according to Figure 5.5. Multiple walk length values produce high validity scores, but we choose walk length 20, which corresponds to the second highest peak. Because using walk length 20 generates cleaner Murofet clusters than a walk length smaller than 10, due to the fact that longer walk length provides more samples of neighborhoods and the model is learned better. The number of walks per node, dimensions, and SGD epoch does not show much difference. We decide on 15 walks, 60 dimensions, and one learning epoch after manual inspection. Lastly, we use a uniform

random probability to choose the next node in the random walk process.

5.5.2 Targeted Noise Injection

We run our version of Pleiades to generate all attacker graphs. Four DGA families were identified: Pykspa, Suppobox, Murofet, and Gimemo. For each we extract the attacker graphs (G) and the ***target domains*** (V). These domains are labeled using the classifier from Section 5.4.1. Before and after the attack, there can be multiple clusters formed within G and G', depending on the graph clustering technique. We use the classifier model to test how likely it is that each cluster belongs to the true DGA malware family, both before and after the attack. We present the overall distribution of the *predicted class probabilities* to show the impact of the attacks.

We use different types of noisy domains at different knowledge levels. For a DGA, these nodes are new NXDOMAINs (V') that will be classified as benign, also queried by the infected hosts (U). In the minimal knowledge case, we create a DGA algorithm that is classified as benign. It is a dictionary DGA that uses the most popular English words from movie subtitles [129], popular web terms, and the top one million Alexa domains. We randomly add numbers and dashes, and randomly select a top-level domain from four choices. In addition, we generate some punycode domains that start with the character sequence "xn–", and some domains with a "www" subdomain. We generate 59,730 verified NXDOMAINs. In the perfect and moderate knowledge cases, the adversary uses existing, unpopular NXDOMAINs from $\mathcal{G}$ and the surrogate dataset, respectively.

Spectral Clustering

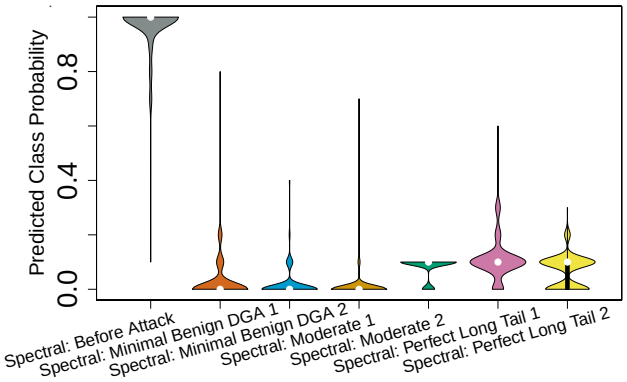
Figure 5.6a shows the classifier's predicted true class probabilities from before the attack is mounted, and after the minimal, moderate, and perfect knowledge targeted noise injection attacks are performed. For each knowledge level, we inject two different levels of noise as described in Section 5.5.2 and re-run the clustering and subsequent classification to assess

the damage from the targeted noise injection. Recall that we try two attack variants, attack variant 1 and 2, where we inject one or two mirrored sets of vertices and edges, respectively. This is to both i) understand how much noise is needed to yield successful evasion, and ii) determine the cost incurred by adding noise.

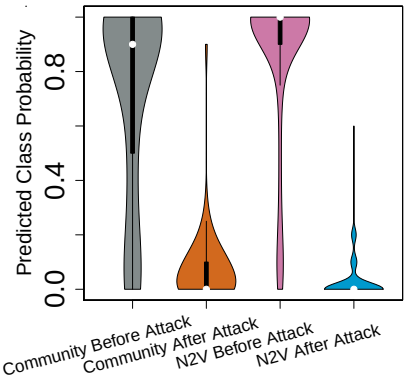
Spectral clustering generates 267 DGA clusters from the four malware families across 12 days. Before the attack, only 0.4% clusters (1 out of 267) are predicted with the wrong labels. In comparison, after the attacks, all clusters are predicted with the wrong labels. Next, we will examine the predicted class probabilities change in the true class label.

Figure 5.6a uses the violin plots to show the distribution of predicted class probabilities for the *true* DGA families, before and after the attacks. The circle is the median value, the thick line within the violin indicates interquartile range, and the thin line depicts a 95% confidence interval. The shape of the violin shows the distribution of the probability values. For example, the first violin in Figure 5.6a has a median of 100% predicted class probabilities, and all data points in the interquartile range have 100% probability value. Specifically, before the attacks, 238 clusters are predicted with 100% class probability that they belong to the true class, and only 28 clusters have a probability between 60% and 100%. For example, the Pykspa cluster had a class probability of only 10% because it contained only two domain names that had very different feature distributions from the majority of Pykspa clusters. The two variants of the attack introduced *at least* 50% and 66% noise to the DGA clusters.

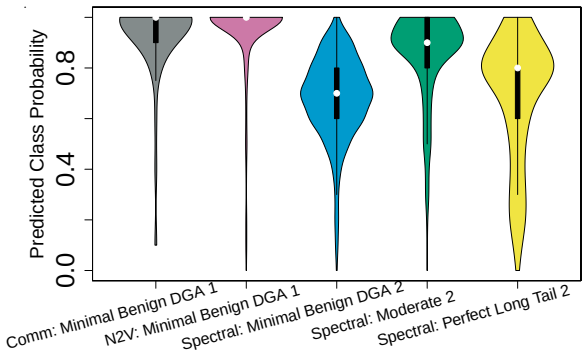
Minimal Knowledge After the attacks, we classify each new adversarial cluster containing target domains and plot the target class probability distributions in the Figure 5.6a. Attack variant 1 ("Minimal Benign DGA 1") generated new clusters with $\leq 80\%$ predicted class probability, with a median of 0%. The predicted class probabilities of 84% of the new clusters drop to zero. Attack variant 2 further decreases the classifier prediction confidence, as shown by "Minimal Benign DGA 2" in Figure 5.6a. After injecting two benign DGA



(a) Spectral Clustering: Predicted class probabilities.



(b) Community Discovery and node2vec: Predicted class probabilities.



(c) Retraining: Predicted class probabilities.

Figure 5.6: Figure 5.6a: Predicted class probabilities before the targeted noise injection attack and after two variants of the targeted noise injection attack in minimal, moderate, and perfect knowledge. Figure 5.6b: Predicted class probabilities before and after the targeted noise injection attacks for community discovery and node2vec. Figure 5.6c: Predicted class probabilities under different attacks after retraining including the "Minimal Benign DGA 1" clusters.

domains, the predicted class probabilities of 87% of the new clusters plummet to 0%. The overall distribution of prediction confidences also shifts downward compared to "Minimal Benign DGA 1".

Perfect Knowledge The median of predicted class probabilities for DGA malware families drops to 10%. As depicted by "Perfect Long Tail 1" in Figure 5.6a, 86% of adversarial clusters were assigned the probabilities of belonging to the true DGA class that are at most

10% . The distribution of class probability values has a smaller variance compared to those in the "Minimal Benign DGA 1". "Perfect Long Tail 2" in Figure 5.6a shows that the maximum prediction confidence is 30%, slightly lower than the maximum 40% confidence from the targeted noise injection attack of "Minimal Benign DGA 2".

Moderate Knowledge We see similar results for the two targeted noise injection attack variants in the moderate knowledge case as in the other cases: a strong drop in predicted class probabilities, with a smaller, more compact distribution of values for attack variant 2. After attack variant 1, 98.3% of new clusters were assigned less than 20% confidence; after attack variant 2, 98.8% of new clusters have less than 20% confidence.

Spectral clustering can be largely defeated at all knowledge levels using the targeted noise injection attacks.

Since previous experiments show that minimal knowledge attackers can carry out targeted noise injection as effectively as more powerful attackers, we will simply demonstrate that the same targeted noise injection attack variant 1 in minimal knowledge also works with community discovery and node2vec.

Community Discovery

We use the same set of DGA domains labeled in Spectral Clustering for evaluation. Before the attack, 80% clusters can be predicted with the correct label, which dropped to 2% after the attack. Figure 5.6b shows the predicted class probabilities for communities containing all *target domains* before and after the attack. Before the attack, the median of predicted probabilities is 90%, and the interquartile range is from 50% to 100%. Specifically, 71 communities contain target domains, among which ten communities only contain one target domain, and seven communities have between 40% to 70% target domains. These noisy communities formed the lower part of the distribution, with $\leq 50\%$ predicted class probabilities in "Community Before Attack", as shown in Figure 5.6b. After the attack, the

Table 5.2: Anomaly cost as percentile of the distinct number of NXDOMAINs queried by hosts, before and after the attack. Only 9.12% of infected hosts become more suspicious, while the rest remain the same.

Before Attack	**$<$ 95th Percentile, 9.12% of hosts**	
Average Increase	From Percentile	To Percentile
Attack Variant 1	69.86%	88.73%
Attack Variant 2	69.86%	93.98%
Before Attack	**$\geq$ 95th Percentile, 90.88% of hosts**	
Average Increase	From Percentile	To Percentile
Attack Variant 1	99.74%	99.85%
Attack Variant 2	99.74%	99.88%

median class probability craters to 0%. Overall 98% of new communities were predicted with lower than 50% probability of belonging to the true class, and 86% of communities have lower than 10% class probabilities.

This demonstrates that the targeted noise injection attack is also effective against the community discovery algorithm.

node2vec

Using the same set of DGA domains labeled in Spectral Clustering, before the attack, 89% clusters can be predicted with the correct label, which dropped to 0.8% after the attack. Figure 5.6b shows that, before the attack on node2vec, the median of predicted probabilities is 100%, and the interquartile range is from 90% to 100%. A total of 85% of clusters were predicted with at least 70% class probability. After the attack, 92% clusters have at most 10% predicted class probabilities.

Targeted noise injection attack also evades node2vec embeddings.

Targeted Noise Injection Costs

It is simple for malware to query additional domains, however, infected hosts engaging in such queries may become more suspicious and easier to detect due to the extra network signal they produce. This may cause the anomaly cost of the targeted noise injection attack

to be high enough to render it useless.

We analyze the anomaly cost by measuring the infected host percentile of the NXDOMAIN distribution both before and after the attacks for the two variants of the targeted noise injection attacks, summarized in Table 5.2. Before any attack, only 9.12% of infected hosts ranked lower than 95^{th} percentile, and the remaining 90.88% of them ranked higher than 95^{th} percentile. This means that, without any attack, infected hosts were already querying more unique NXDOMAINs than most hosts in the network. However, doing targeted noise injection attacks further increases the percentile ranks of the infected hosts, but not substantially.

We separated the results based on whether infected hosts were querying fewer domains than 95% of all hosts in the local network. Table 5.2 shows that among the 9.12% infected hosts ranked lower than 95^{th} percentile before the attack, they increased from an average percentile of 69.86% to 88.73% after the targeted noise injection attack variant 1. Furthermore, they increased to 93.98% after attack variant 2. However, 90.88% of infected hosts did not become more anomalous. They were ranked higher than the 95^{th} percentile before the attack. Their average percentile increased by 0.11% after attack variant 1, and by 0.14% after attack variant 2. Because they were querying more domains than other hosts before the attack, injecting noise does not change their percentile substantially.

The majority of hosts had little change in "suspiciousness", whereas a small percentage of hosts increased their suspiciousness after the targeted noise injection attacks.

5.5.3 Small Community

We choose a group of 618 domains and 10 infected hosts belonging to Suppobox as the basis for the small community attack. They form a community using the community discovery algorithm, and two clusters using spectral or node2vec embeddings. A small community attack is successful if and only if *all* DGA domains join either the "death star," or clusters where the subsequent classifier does not predict them as the true malware DGA

class. Recall that the death star cluster contains tens of thousands of domains that cannot be properly classified. In all experiments, the small community plots denote the configurations where an attack succeeds based on the aforementioned criteria. This is represented by green regions (see Figure 5.7) when the "death star" is joined, or white cells when the noisy clusters cannot be predicted as the true class label (see Figure 5.9) when using node2vec.

Spectral Clustering

As described earlier, the small community attack can only be verified in the perfect and moderate knowledge cases. In the minimal knowledge case, however, an attacker can still mount the attack by randomly removing edges and nodes, as described in Section 5.2, while hoping for the best.

Minimal Knowledge The upper-leftmost plot in Figure 5.7 demonstrates the possible successful configurations for mounting the small community attack by randomly removing nodes and edges. The plot shows the remaining number of NXDOMAINs on the Y-axis ($|V| - n_v$) and the remaining number of connections from infected hosts for each NXDOMAIN on the X-axis ($|U| - n_e$). The shaded region shows approximately a 75.16% success rate for an attacker with no knowledge of the defender's graph $\mathcal{G}$. While a minimal knowledge attacker cannot guarantee their attack will succeed, they nonetheless have a high chance of success.

Perfect Knowledge The upper-left plot in Figure 5.7 depicts a successful small community attack area when the computed SVD rank is 35. The figure shows that only a small set of configurations with 380 to 618 DGA domains, each queried by between 3 to 10 random hosts, were unable to successfully launch a small community attack. The cost of the small community attack is very low against the system, which is configured with rank 35 and runs in this network. For example, an adversary controlling the DGA does not need to give up any infection, but only needs to reduce the number of infected hosts that query

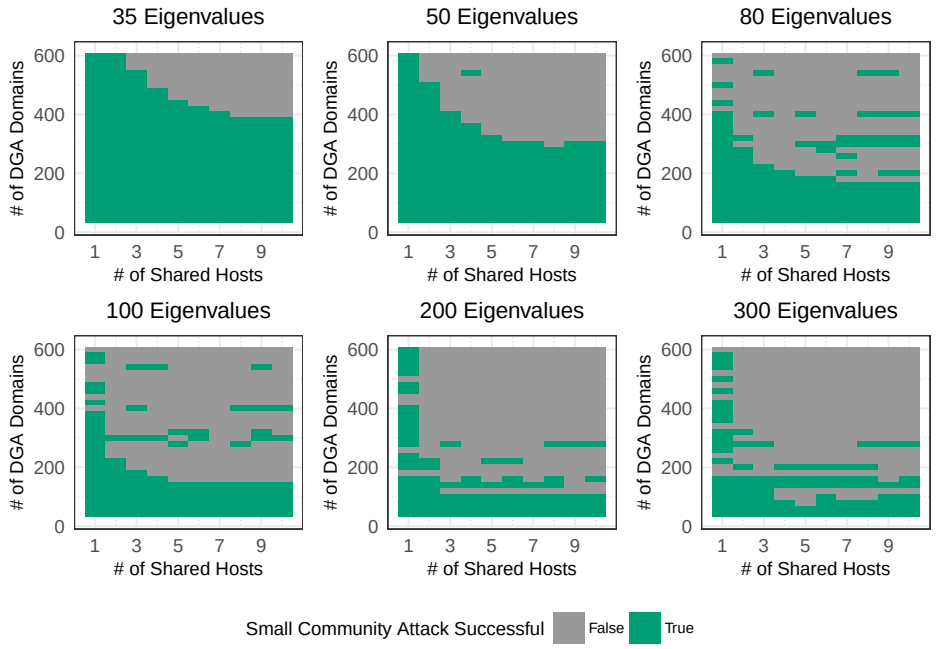


Figure 5.7: Different number of eigenvalues.

a common set of DGA domains from 618 to 380 in order to hide the domains. In this case, by removing $n_v = 238$ NXDOMAINs, the attacker does not lose any additional host querying activities $min(n_e) = 0$. But if the attacker needs extra redundancy provided by 460 distinct NXDOMAINs, each domain can only be queried by a subset of 5 hosts. Then $n_v = 158$, and accordingly, $min(n_e) = 5$. In this case, the attacker does not need to lose control of any infected hosts, but she does need to coordinate each five infected hosts to query a subset of distinct NXDOMAINs that do not overlap with each other.

Moderate Knowledge After reducing the number of DGA domains and the number of infected hosts per domain to the successful attack area shown in Figure 5.8, the DGA domains join the surrogate death star. We test that these values also work to join the original death star. Because the original network size is larger than the surrogate network size, the real successful area (the top-left plot in Figure 5.7) is much bigger than the one shown Figure 5.8. Thus, the small community attack works with moderate knowledge when the surrogate dataset is a smaller network than the original network. If the surrogate dataset is

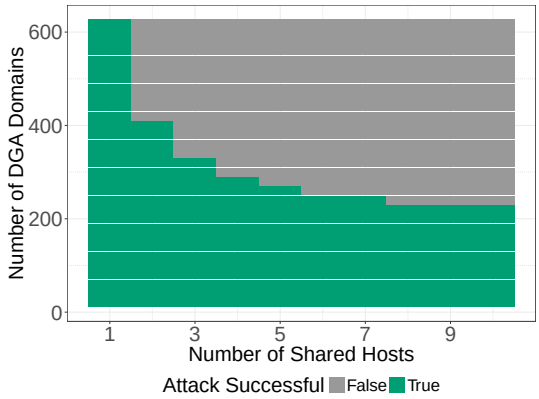


Figure 5.8: Success area for joining the death star of the surrogate dataset in the moderate knowledge case. All the successful attack configurations worked in the ground truth network.

a larger network, the adversary may miscalculate the cost of joining the death star, which may not work in the original network. By using such a surrogate dataset, the adversary will likely choose fewer DGA domains and their shared hosts to simulate a successful attack, compared to the ideal case in perfect knowledge. In other words, the practical cost of launching a small community attack with moderate knowledge is more than the minimal cost of such an attack in the original network. We explore the effect of network size in Section 5.5.3.

Spectral clustering can be evaded using the small community attack, even when the attack cannot be verified by the attacker with a success rate of 75%+. More sophisticated attackers can always evade.

Community Discovery

Unlike graph embedding techniques that lose information about smaller components of the graph, community discovery algorithms do not lose information and can properly handle portions of $\mathcal{G}$ with exactly one edge. Rather than clustering poorly with other small components, they are considered to be separate communities. So the cost of the small community attack is much higher than with graph embeddings because attackers must generate singletons that are small enough to evade classification, forcing the attacker's graph to be

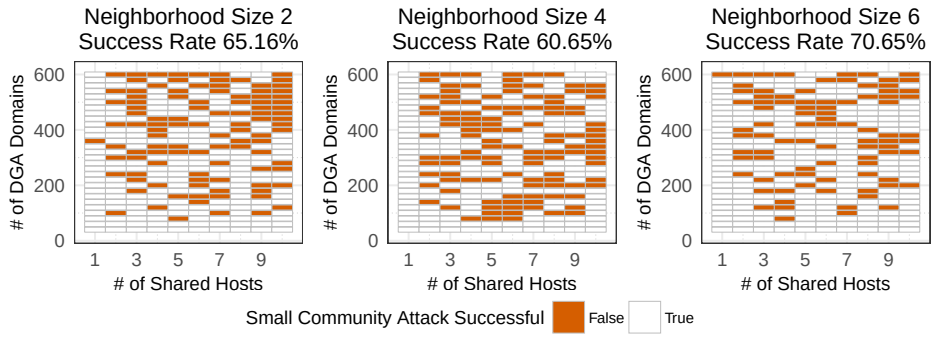


Figure 5.9: Success area of small community attacks with different context size.

disconnected. Therefore, they can evade clustering with the cost of losing their ability to efficiently manage their bots. For example, to evade community discovery in the example presented in Figure 5.2, an attacker would have to use the modified attack graph G'_4 and the drop from $D(G) = 0.5$ to $D(G'_4) = 0.25$ is enough to consider the attack cost too high. In the DGA case, this would mean each infection would need its own distinct domain-name generation algorithm, which would be an exceedingly high cost for an attacker. As such, we do not compute results for small community attacks on community discovery.

Community discovery is resistant to the small community attack due to the high costs it would cause the attacker, however, spectral methods and node2vec are more likely to be used by defenders as they result in cleaner clusters and better classification results.

node2vec

The third plot in Figure 5.9 shows that the small community attack is still possible with node2vec, using aforementioned hyperparameters (Section 5.5.1). The attack is possible when the number of shared hosts is 1 (the first column except the top cell), and when the number of DGA domains is ≤ 40 (the bottom two rows). Elsewhere, the attack succeeds randomly due to the random walk. In summary, the small community attack is definitely possible with very small component sizes. Compared to SVD, the cost is higher here. For example, the attacker needs to give up $n_v = 578$ unique NXDOMAINs in a day, along with $n_e = 0$, for the small community attack to be successful. But if the attacker is not

willing to give up such cost, the small community attack is not guaranteed to succeed given the randomness of neighborhood sampling. However, if a minimal knowledge attacker randomly chooses any n_v and n_e for a small community attack, she will have a 70.65% attack success rate shown by the third plot in Figure 5.9.

node2vec is susceptible to the small community attack, but with fewer guarantees and higher costs than in the spectral case, due to its inherent randomness. node2vec being used in Pleiades would render the system more resilient against small community attacks.

Small Community Costs

The cost of the small community attack is affected by both the size of network and change in density when the attack is performed.

Size of Network The network size is related to the number of nodes (hosts and domains) and the number of edges (the query relationship). As a straightforward way to model the network size, we randomly sample the hosts in the ground truth network dataset along with all domains queried. We also keep the same attacker subgraph G, containing the Suppobox DGA community with 10 infected hosts and 618 DGA domains, along with other domains queried by these hosts for the experiment.

Figure 5.10 shows the small community attack results by sampling 10% to 90% of all hosts. When only 10% of hosts were sampled, the small community attack failed in most areas of the plot. The attack success area increases as the network size gets larger. This means that the cost for small community attack is lower in a larger network than in a smaller network, given the same hyperparameters. A larger network is harder to accurately represent in an embedding, which provides more areas for attackers to hide and evade.

A moderate knowledge level attacker should attempt to acquire a surrogate network dataset smaller than the ground truth network dataset for a safe estimate of their small community attack cost.

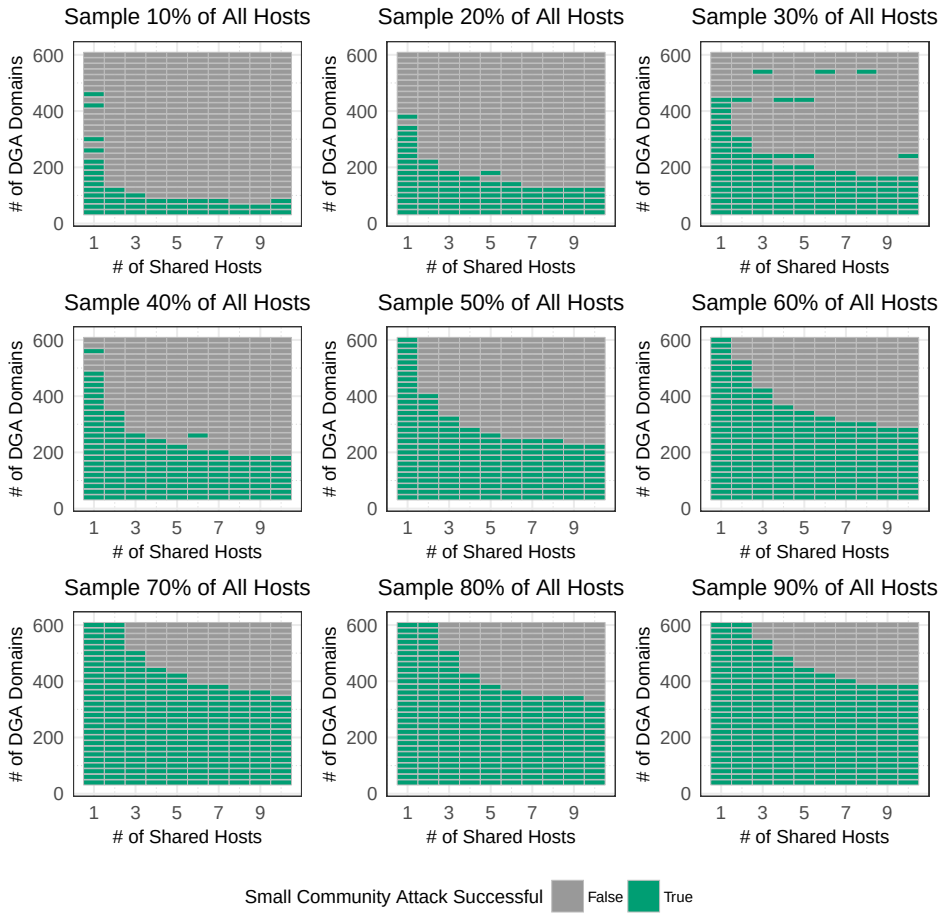


Figure 5.10: Different sizes of the network dataset.

Agility Cost By removing nodes and edges, the attacker loses redundancy. For example, hosts need to query fewer DGA domains, or malware can be allowed fewer malicious actions. We measure the agility cost by the change in density of the attack graph. Density captures the number of edges present in the attack graph over the maximal number of possible edges. In Section 5.5.3, Equation 5.1 defines the attack graph density before the small community attack; and Equation 5.2 defines the density after the attack. Before the attack, $D(G) = 0.48$ for the Suppobox community. For each SVD rank parameter, we record attack configurations that were successful small community attacks as green ares in Figure 5.7. There are some outliers outside the continuous area. Although these attacks do not make NXDOMAINs join the death star, they move NXDOMAINs to clusters that cannot be predicted with the correct label. To measure the minimum agility cost, we

Table 5.3: Agility cost of small community attacks under different hyperparameter configurations.

	Spectral Clustering		
	Density		
Join Death Star	Median	Maximum	Minimum Cost
SVD rank 35	0.078	0.61	0
SVD rank 50	0.11	0.45	0.03
SVD rank 80	0.065	0.26	0.22
SVD rank 100	0.052	0.19	0.29
SVD rank 200	0.0032	0.10	0.38
SVD rank 300	0.026	0.26	0.22
	node2vec		
	Density		
		Maximum	Minimum Cost
Neighborhood Size 6		0.065	0.415
Number of Walks 5		0.065	0.415
Number of Walks 10		0.032	0.45
Number of Walks 15		0.065	0.415
Walk Length 2		0.65	0
Walk Length 4		0.29	0.19
Walk Length 12		0.065	0.415
Walk Length 20		0.065	0.415

exclude the outliers by only calculating attacker graph density that resulted in joining the death star. Table 5.3 summarizes the median and maximum attacker graph density in these small community attacks, with the minimum cost represented by the difference between $D(G)$ and $max(D(G'))$. When the SVD rank is 35, the $max(D(G'))$ to join the death star is slightly bigger than $D(G)$, which means there is no cost in launching the small community attack. In this case, the attacker can evade while having *more* connectivity. As the SVD rank increases, the attacker graph density is reduced, which means a successful attack is more costly to the adversary. Also, the minimum cost increases as the SVD rank increases. For example, when the SVD rank is 80, the minimum cost is 0.22, reducing the attack graph density from 0.48 to 0.26. The attacker needs to reduce the number of distinct DGA domains from 618 to 160 to evade, but each domain can be queried by all infected hosts. In comparison, when the SVD rank is 200, the minimum cost is 0.38. The attacker needs to further reduce the number of distinct DGA domains to 60 to evade, with

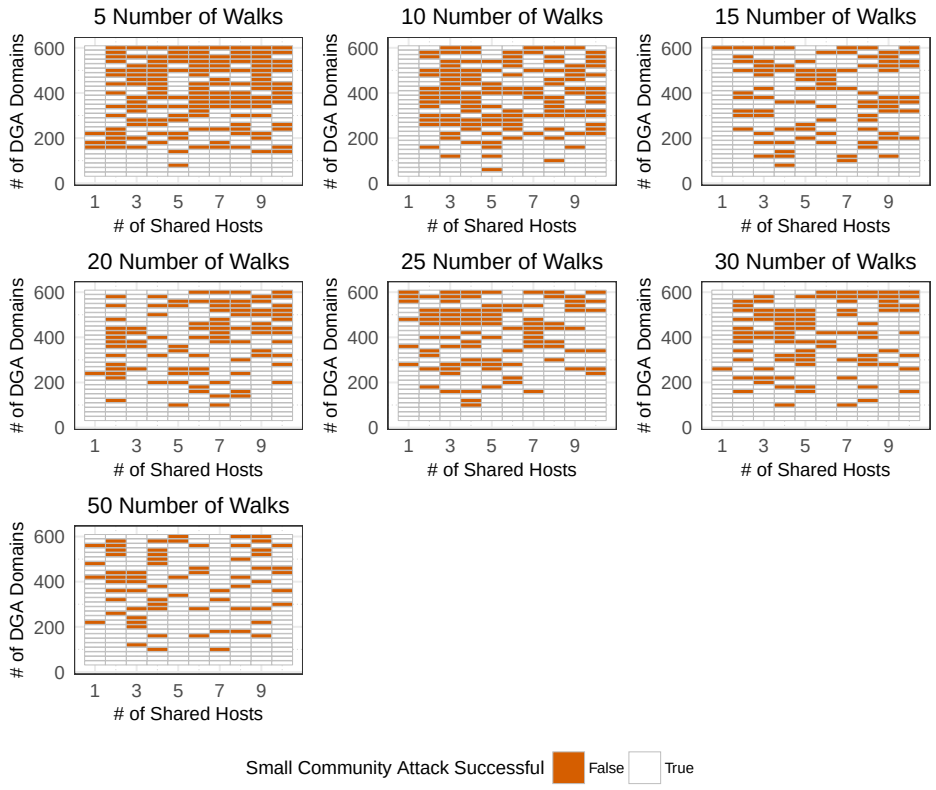


Figure 5.11: Success area of small community attacks with different number of walks.

each domain queried by all infected hosts. The attack graph density is reduced from 0.48 to 0.1, losing 79% ($\frac{0.38}{0.48}$) of queries to distinct DGA domains. This means that tuning hyperparameters can increase the small community attack cost and potentially render this attack ineffective.

Similarly, for node2vec, the minimum cost of a certain small community attack is higher than spectral clustering. We compute the attacker graph density only for the white area in Figure 5.9 without randomness, i.e., the first column and bottom two rows. In contrast to spectral clustering, node2vec requires a much higher minimum cost for a guaranteed small community attack, which indicates that node2vec is more resilient to this attack. The smallest communities in Figure 5.9 (i.e., the first column and bottom two rows) are likely undersampled, because choosing 15 walks per node and walk length 20 using cluster validity in Section 5.5.1 prefers labeled DGA communities that are relatively bigger, which makes few neighborhood observations for extremely small islands insignificant, and thus

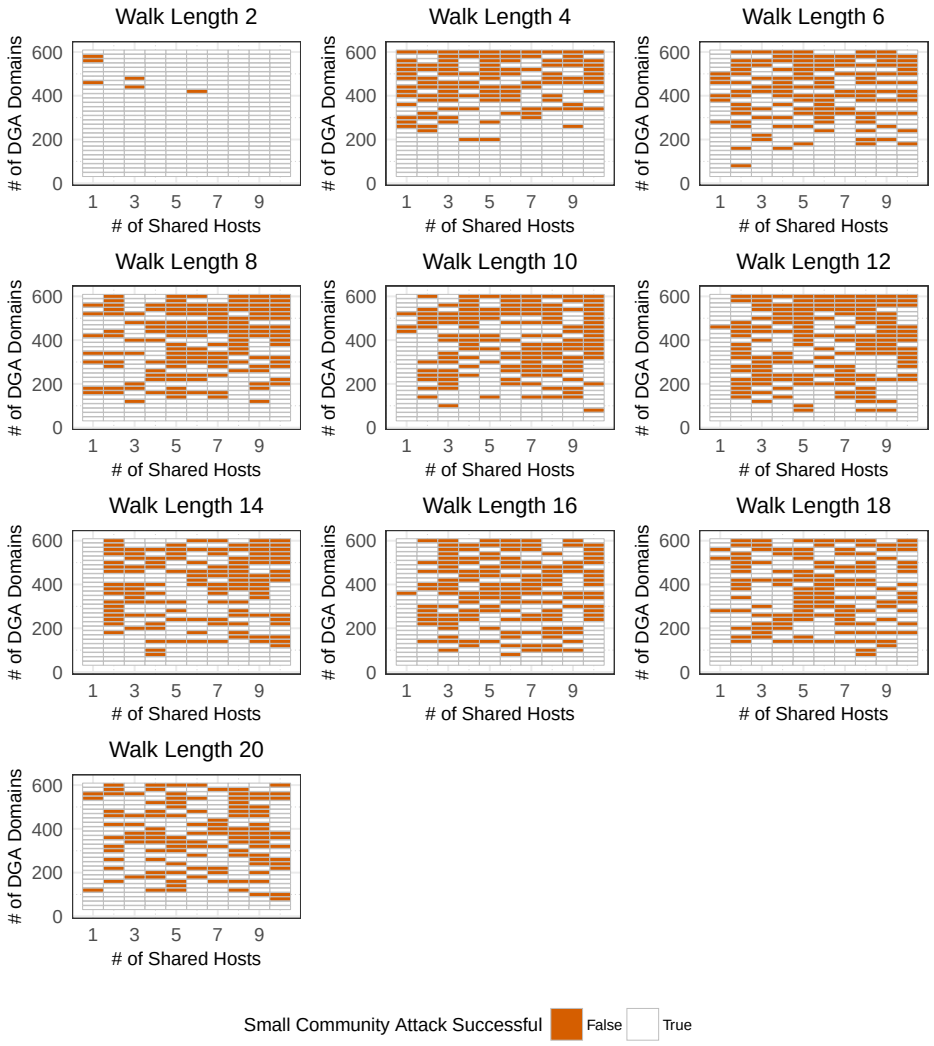


Figure 5.12: Success area of small community attacks with different walk length.

allows small community attacks. Note the randomness in the remaining portion of the plot. Since node2vec uses the random walk process to sample the neighborhoods of all nodes, there exists randomness in the neighborhood observations. This shows that the randomness inherent to node2vec makes the attacks succeed at random in the remaining portion of Figure 5.9. This both suggests a system like Pleiades would benefit from node2vec to reduce the guarantee of attacks, as well as allow a defender to identify if an attacker is evading by chance encounters where the evasion fails over time. While the minimum attack cost is the same with different neighborhood sizes for a guaranteed successful attack, the attack

success rate changes. The neighborhood sizes 2, 4, and 6 have attack success rate 65.16%, 60.65%, and 70.65% respectively (Figure 5.9). We will discuss how we can use different hyperparameters to further reduce the success rate of the small community attack in Section 5.6.2.

Despite the differences in the attack success rate, different neighborhood sizes in our experiments have the same minimum cost for the small community attack. In comparison, number of walks has slightly different minimum cost over different parameter values. As shown in Figure 5.11 and Table 5.3, the minimum cost for guaranteed small community attack under different number of walks are 0.415 or 0.45. However, Figure 5.12 and Table 5.3 show that different values for the walk length parameter have different minimum attack cost. When the walk length is 2, there is no attack cost; but when the walk length is 20 (the parameter we choose by using cluster validity), the minimum cost is 0.415. In Section 5.6.2, we will show that walk length 2 and 4 have higher small community attack success rate than longer walk lengths.

These costs further demonstrate node2vec's superiority over spectral clustering in resisting small community attacks.

5.6 Defense

Since the noise injection attack and the small community attack violate the fundamental assumptions used by graph clustering techniques, it is very hard to completely eliminate the problem. In this section, we propose two defense techniques that help Pleiades retain its detection capabilities against the two attacks. The first one is to train the classifier with noise, which remediates the noise injection attack to some extent. The second one is to use the small community attack as an adversarial guideline to choose better hyperparameters for graph embeddings, which increases the cost of launching a successful small community attack.

Table 5.4: False Positive Rate for four DGA families before retraining, and after retraining with three types of noise.

	False Positive Rate			
Model	Pykspa	Gimemo	Suppobox	Murofet
Original	0.32%	0.29%	0%	0%
Model A	1.64%	0.39%	0.10%	0%
Model B	1.62%	0.10%	1.23%	0.30%
Model C	1.46%	1.17%	1.23%	0%

5.6.1 Training Classifier with Noise

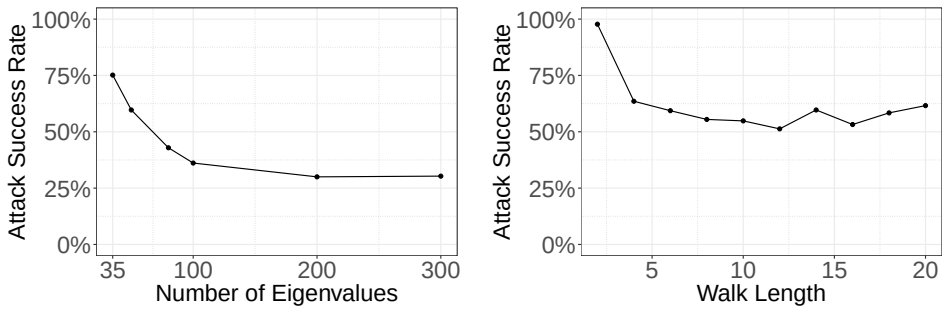
By retraining the classifier, it becomes more resistant to noise that could be injected by the adversary in the unsupervised phase of Pleiades. We used domains from the benign DGA to poison the clusters of malicious DGAs. We retrained the classifier using clusters generated by the noise injection attack variant 1 ("Minimal Benign DGA 1", $m = 1$, Algorithm 2 in Section 5.5.2) from SVD, yielding *model A*. We tested model A against the adversarial clusters generated by the same noise injection attack under community discovery and node2vec. The first two violins in Figure 5.6c show that model A increases the overall predicted class probabilities compared to the "After Attack" violins in Figure 5.6a. In community discovery, the accuracy increased from 2% to 98%; and in node2vec, the accuracy increased from 0.8% to 98%. To summarize, retraining with noisy clusters containing a benign DGA from SVD can remediate the same attack on community discovery and node2vec. We see this same effect even when the noise levels are doubled ($m = 2$, Algorithm 2 in Section 5.5.2). When models were trained with half the noise ($m = 1$, Algorithm 2 in Section 5.5.2), they were able to more accurately predict the correct label. Among them, only an average of 7.3% clusters are predicted with the wrong labels, decreased from 100% before retraining.

In comparison with Figure 5.6a, the average prediction confidence increased significantly. Before retraining, the average prediction confidence of "Minimal Benign DGA 2", "Moderate 2", and "Perfect Long Tail 2" are 10%, 20%, and 20%. After retraining, they

increased to 70%, 90%, and 80%, respectively. The accuracy of the models remain roughly the same before and after retraining. However, retraining with noisy clusters increased the false positive rate in most cases (Table 5.4).

It is important to note that this defense only trains the classifier with noise that has been witnessed. New noise will appear, but the fundamental attack on the unsupervised component remains the same. Therefore, defenders will be alerted by plumetting accuracies in their models. Our defenses are simple and future work should be done to make clustering systems more robust.

5.6.2 Improving Hyperparameter Selection



(a) (b)

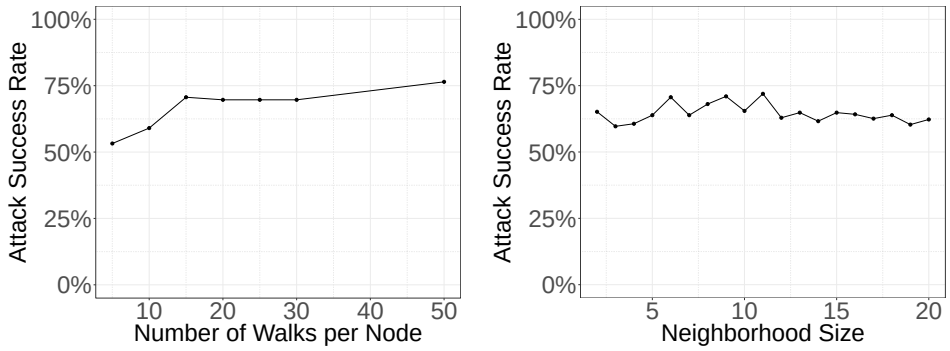
Figure 5.13: Figure 5.13a: Using the small community attack to choose the number of eigenvalues for SVD. Figure 5.13b: Using the small community attack to choose the length of walk for node2vec.

Small community attacks show that the traditional ways of choosing hyperparameters (Section 5.5.1) is not enough when facing adversaries. Luckily, the small community attack can be used to choose more resistant hyperparameters. We show that better selection can reduce the number of successful small community attack instances from our previous experiments.

We plot the successful attack rate under different number of eigenvalues in Figure 5.13a. The successful attack rate decreases as the number of eigenvalues computed increases, and the line plateaus after 200 eigenvalues. It means that a defender running Pleiades should

select the first 200 eigenvalues, instead of 35 indicated by the scree plot in Figure 5.4. If we use the small community attack in this way, we can choose better parameters for the system and also know under which parameters the system is vulnerable.

Similarly, for node2vec, using the small community attack to choose hyperparameters can reduce the attack success rate. The cluster validity metrics suggest we choose neighborhood size six, and walk length of 20. However, if we evaluate the graph clustering using the success rate of the small community attack, these hyperparameters are not optimal. First, for the neighborhood size, Figure 5.9 shows that a smaller neighborhood size of four introduces a lower attack success rate. Second, we plot the attack success rate under different walk lengths in Figure 5.13b. This figure shows that a walk length of 12 is preferred over 20, because the former allows 51.29% attack success rate compared to 61.61% of the latter. In other words, the smaller neighborhood size and shorter walk length can tolerate the small community attack better, presumably because they do not oversample larger communities with more distinct neighborhood observations. In other words, smaller communities are not undersampled.



(a) (b)

Figure 5.14: Figure 5.14a: Using the small community attack to choose the number of walks per node for node2vec. Figure 5.14b: Using the small community attack to choose the neighborhood size for node2vec.

In addition, two more parameters of node2vec can be tuned by using the small community attack. Figure 5.14a shows the sensitivity of parameter, different number of walks per node, to the small community attack. Under 15 walks per node, the small community

attack success rate is 70.65%. Overall, the attack success rate increases as the number of walks increases. If we want the minimal small community attack success rate, we should choose smaller number of walks per node, such as 5 or 10. Using 5 walks per node can reduce the attack success rate to 53.23%. Similar to our results of tuning the walk length, a smaller value for number of walks prefer small communities, but a larger value undersamples small communities. Lastly, different values of the neighborhood size parameter have slightly different attack success rate, as shown in Figure 5.14b. Overall, from neighborhood size 2 to 20, the attack success rate oscillates between 59.68% and 71.94%. However, the neighborhood size is less sensitive to the small community attack than the walk length and the number of walks.

We recommend using the small community attack success rate to evaluate the clustering hyperparameter selection, in addition to traditional cluster validity indices.

5.7 Summary

We have demonstrated that generic attacks on graphs can break a real-world system that uses several popular graph-based modeling techniques. These attacks can often be performed by limited adversaries at low cost; however, simple defenses can reduce their effectiveness or likelihood of success. To summarize how defenders can improve their systems: hyperparameter selection should be optimized for reducing the success rate of small community attacks, and retraining can be used to lessen the impact of noise injection attacks. Furthermore, state of the art graph embedding techniques like node2vec appear to be more resistant against small community attacks, which suggests Pleiades and other systems would be harder to adversarially manipulate using node2vec over community finding, or spectral methods (see Figure 5.9 vs. Figure 5.7).

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Overall Contribution and Summary

This thesis focuses on two goals, DNS graph clustering applications and improving the robustness of clustering. For the first goal, we propose new clustering techniques to solve two long-term advertising fraud problems. For the second goal, we propose a threat model to evaluate how attackers with different knowledge levels can evade clustering. We propose two novel yet simple attacks. Through the adversarial clustering analysis, we can identify where the clustering system is vulnerable, and how we can improve the robustness of the system.

6.2 Future Work

We acknowledge that details surrounding the implementation of our adversarial clustering attacks are specific to Pleiades, however, the graph representation suggests the attacks may work on other graph-based systems. In this section, we briefly discuss issues to consider to generalize the attacks, as well as future work direction.

In other types of security graphs, an attacker needs to perform different actions in order to manipulate nodes and edges in the attacker graph. Overall, an attacker needs to associate her nodes with benign ones for the targeted noise injection attack, and reduce overlap of her nodes' behavior for the small community attack. For instance, in the bipartite graph of machines (U) and md5s of installed executables (V) [130], malware authors can bundle malicious executables with benign ones via Pay-Per-Install (PPI) programs, in order to inject noise; or they can program the installed malware to write to different md5s for different machines, and delete their old malware, in order to small communities. As another

example, in the malware behavioral graph between the binary (U) and its behavioral profile or Malware Instruction Set (V) [123, 131, 27], the malware authors can change the executable's system call frequency and sequence to approximate what benign software does, in order to inject noise; or they can make system call sequence as distinct permutations for different machines, in order to create small communities.

Nodes and edges can be trivially injected or removed in the graph Pleiades uses, which are generated by malware resolving domain names. In other security contexts, the set of injectable/removable nodes varies. It is possible that some nodes and edges must exist in order for certain attack actions to succeed. For example, a phishing email using a malicious attachment requires at least the *read* system call to successfully infect a host, which cannot be removed from the system call graph. On the other hand, it can be difficult to add certain nodes and edges. Therefore, in addition to the anomaly cost (Section 5.5.2) and agility cost (Section 5.5.3), the action of graph manipulation itself has costs depending on the data that underlies the graph representation. This should be carefully considered when generalizing the attacks to other systems. Tighter costs may exist, but our approaches point in a promising direction.

Results from our adversarial clustering analysis show that the small community attack is not guaranteed to succeed if an adversary does not have perfect knowledge about other nodes on the entire graph, or the clustering system parameters. Since the defender running the clustering system is essentially using global features, it provides an advantage against adversaries. On the other hand, it is beneficial to choose global features over local features for classification tasks as well. For instance, pixels from an image, words in an email, and metadata of PDFs are local features, which can be easily manipulated by adversaries. On the contrary, if features can be chosen that requires knowledge about other images, emails, PDFs that an attacker does not have easy access to, it makes the classifier more robust. For example, in domain name reputation systems, the set of related historical IP addresses resolved by domain names over a time window (e.g., past six months) is a global feature.

If an evasive attacker changes domain name resolution at the moment of detection, it does not change the feature easily because resolutions from the past are still used by the feature. Guidelines for how to change local features to global features can increase the costs for attackers to evade classifiers.

Appendices

APPENDIX A

DGA DETECTION

A.1 Unique Domains queried by Hosts

Figure A.1 shows the cumulative distribution for distinct number of NXDOMAINs queried by hosts seen on 12/18/2016 in the network datasets from the telecommunication network. The CDF shows that a host querying two distinct NXDOMAINs is at the 48^{th} percentile, and a host querying 10 distinct NXDOMAINs is at the 95^{th} percentile.

A.2 Labeled DGA Families

We use default parameters to generate different versions of the malware families for 18 different seed dates. The number of domains generated for each malware family is recorded in the top part of Table A.1.

A.3 Reimplementing Pleiades

We implement a simplified version of the Pleiades DGA detection system. We follow the exact next steps to implement the graph clustering and modeling components of Pleiades.

1. From the NXDOMAIN query data, we filter out hosts that only queried one domain name in a day (as the authors of Pleiades did).

2. We construct an association matrix representing the bipartite graph between hosts and the NXDOMAINs they queried. Each row represents one host and each column represents one NXDOMAIN. If host i queried NXDOMAIN j in that day, we assign weight $w_{ij} = 1$ in the matrix. Otherwise, we assign $w_{ij} = 0$. Then, each row is normalized such that the sum of weights is one.

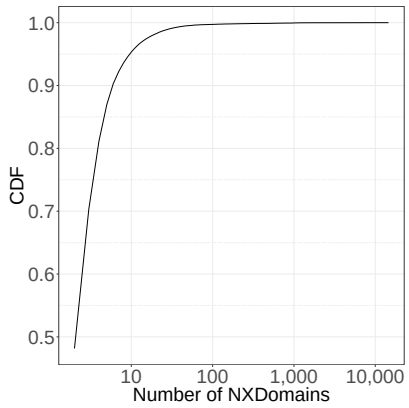


Figure A.1: Cumulative distribution of distinct number of NXDOMAINs queried by each host in 12/18/2016.

3. Next, we do Singular Value Decomposition (SVD) over this matrix and keep the first N eigenvalues. For our dataset, we choose $N = 35$ according to the scree plot of Eigenvalues. Figure 5.4 shows that the Eigenvalues line plateaus after $N >= 35$.

4. The resulting eigenvectors are used for XMeans clustering.

5. Once we have the clusters of NXDOMAINs, we extract a feature vector for each cluster, which will be used for classification. We have four feature families: length, entropy, pairwise jaccard distance of character distribution, and pairwise dice distance of bigram distribution. This yields a 36-length feature vector for classification that relies on properties of the domain strings themselves. Please refer to Section 4.1.1 in the original Pleiades paper [25] for further details.

6. Finally, the classifier uses the feature vectors of the clusters to detect existing, known DGAs and identify never-before-seen DGAs.

To obtain DGA domains as a training dataset for the classifier, we analyzed dynamic malware execution traffic and executed reverse-engineered DGA algorithms. Firstly, we identified NXDOMAINs that were queried by malware md5s by analyzing malware pcaps obtained from a security vendor. We used AVClass [132] to get the malware family labels of those md5s. Using this method, we labeled pykspa, suppobox, and gimemo malware

Table A.1: DGA families contained within our ground truth dataset.

DGA Family	# of Domains	# of Feature Vectors
Chinad	4,608	18
Corebot	720	18
Gozi	864	72
Locky	360	36
Murofet	54,720	56
Necurs	36,864	18
NewGOZ	18,000	18
PadCrypt	1,728	36
Qadars	3,600	18
Qakbot	180,000	35
Ranbyus	720	18
Sisron	739	19
Symmi	1,152	18
Pykspa	90,300	48
Pykspa	1,190	40
Gimemo	9,144	17
Suppobox	12,846	40

families, which were active in our dataset. We extract one feature family per cluster for these. Secondly, we use reverse engineered DGA domains to compensate limited visibility of DGAs active in the network dataset. Although only Pykspa, Suppobox, and Murofet domains have matches in active clusters, we extract one feature vector for each version's daily domains of 14 DGA families from the reversed engineered DGA domain dataset. Table A.1 shows the distribution of the number of features vectors from the reverse engineered DGAs (top) and those seen in clusters (bottom).

We trained the classifier with 17 classes, including 16 malware families and one manually labeled benign class. We labeled benign class from clusters containing mixture of all kinds of benign domains, as well as clusters containing disposable domains (e.g., DNS queries to Anti-Virus online reputation products [133]).

We performed model selection to choose among the following algorithms: Naive Bayes, Linear SVM, Random Forest, Logistic Regression and Stochastic Gradient Descent Classifier. After the analysis of the performance of the different classifiers, we chose to use

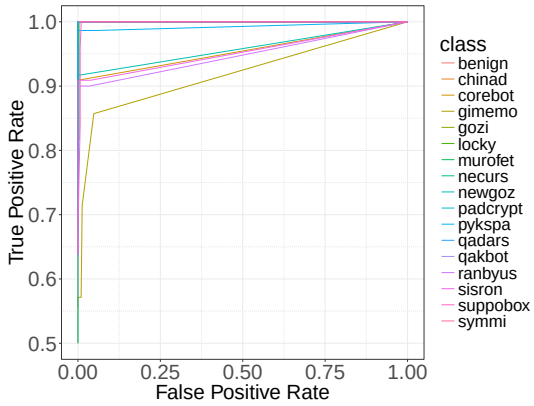


Figure A.2: ROC curves for 16 malware DGA classes and one benign class.

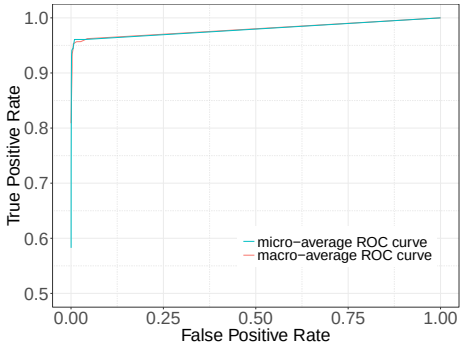


Figure A.3: Micro and macro ROC curves.

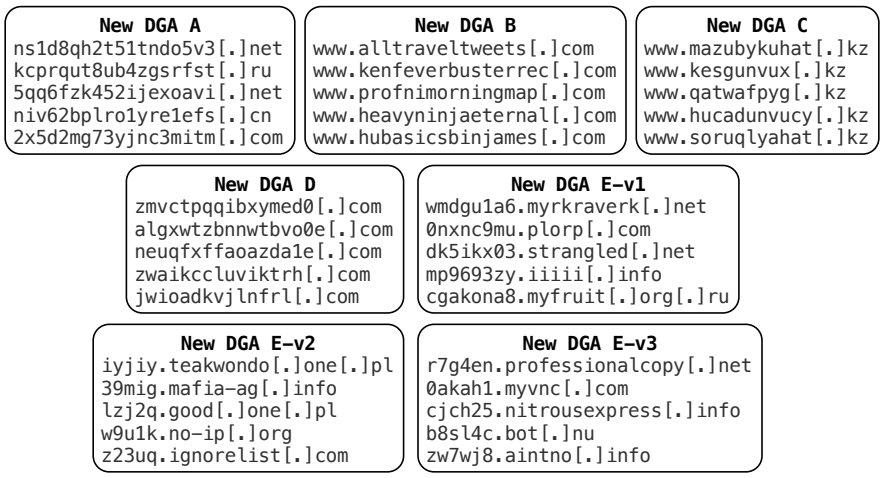


Figure A.4: Newly found DGAs.

Random Forest as our classifier. Random Forests are similar to Alternative Decision Trees, a boosted tree-based classifier, which were used in the original Pleiades paper. We tested our classifier with five fold cross-validation and measured an average accuracy at 96.08%, and a false positive rate of 0.9%. Figure A.2 shows the multi-class ROC curves of the classifier performance. Figure A.3 shows the micro and macro ROC curves of the multi-class classifier in our implementation of Pleiades.

A.4 Current DGA Landscape

We ran the DGA detection system over anonymized network traffic from a Recursive DNS server in a telecommunication provider, from December 18, 2016 to December 29, 2016.

Newly Discovered DGAs We found 12 new DGA malware families. Figure A.4 shows 5 of them. New DGA A is classified as similar to the DGA Chinad, with a total of 59,904 domains. The generated domains have a fixed length of 18 characters and use five different tlds: .com, .net, .cn, .biz, and .ru. Chinad has similar characteristics in domain names, but its domain length is 16 characters, and it uses two additional tlds: .info and .org. New DGA B is a dictionary-words DGA that is classified as similar to Gozi. Gozi generates domains by combining words from word lists such as Requests for Comments

(RFC), the Ninety Five Theses of Martin Luther in its original Latin text, and GNU General Public License (GNU GPL). In 12 days, we observed 9815 domain names from this DGA, with 10,435 infected hosts. New DGA C is classified as similar to Gimemo. It repeatedly uses bigrams and trigrams as units for composing domain name strings. We found 6,738 domains for new DGA C. Most of the domains from DGA C follow a pattern of consonant-vowel-consonant at the beginning, usually followed by another similar pattern or a sequence of vowel-consonant-vowel, which makes the generated domains appear almost readable. Nevertheless, New DGA C generated domains did not follow the character frequency distribution for any of the languages that use the English or similar alphabets. The length of the generated domains is not fixed but it appears to be around 10 characters with either a character added or removed. New DGA D uses `.com` tld, and second-level labels varying between 12 and 18 characters. New DGA E-v1 iterates through both algorithm-generated second level domains and child labels.

Evasion Attempts in the Wild The DGAs of qakbot and pykspa provide us with evidence that the malware authors are attempting to avoid or obstruct detection. A special mode of Qakbot is triggered when the malware detects that it is running inside a sandbox environment. Specifically, the seed of the algorithm is appended to generate redundant domains that won't be used as actual C&C. Similary, Pykspa generates two sets of domains based on two different seed sets, which appear identical to a human analyst as if there were only one set of generated domains. Different than Qakbot, in normal operation Pykspa queries both sets of domains, along a list of benign domains. This kind of behavior could be a method to detect analysis efforts. If an analyst sets the environment to provide answers to these "bogus" queries, it could indicate anomaly to the malware. Generating a large number of "fake" domains could also increase the cost of sinkholing the botnets. It makes the sinkholing operation more likely to fail to cover all of the actual C&C domains [134]. These efforts appear to be in their infancy in terms of complexity and effectiveness at this

point. If malware authors unleash their creativity in the future, we might come across more elaborate evasion cases that require a lot more effort to identify and detect.

Furthermore, we identify instances of DGAs already evading the classification part of Pleiades by introducing a child label. Our classifier has low confidence for detecting new DGAs B, C, and E-v1. Since there are no DGA domains with child labels in the training dataset, the classifier does not have the requisite knowledge to predict such DGAs. After deploying the classifier for 12 days, we retrained the classifier with additional DGA families observed from the network. After retraining, our classifier has successfully identified the following new variants with high confidence: DGA E-v2 and DGA E-v3.

REFERENCES

[1] ITNews. *Inside eBays 90PB data warehouse*. https://www.itnews.com.au/news/inside-ebay8217s-90pb-data-warehouse-342615.

[2] The Economist. *Data, data everywhere*. http://www.economist.com/node/15557443.

[3] Interactive Advertising Bureau. *Viewability Has Arrived: What You Need To Know To See Through This Sea Change*. http://www.iab.net/iablog/2014/03/viewability-has-arrived-what-you-need-to-know-to-see-through-this-sea-change.html. 2014.

[4] Google. *Just in time for the holidays - viewability across the Google Display Network*. http://adwords.blogspot.co.uk/2013/12/just-in-time-for-holidays-viewability.html. 2013.

[5] FBI New York Field Office. *Press Release: Defendant Charged In Massive Internet Fraud Scheme Extradited From Estonia Appeared In Manhattan Federal Court*. https://archives.fbi.gov/archives/newyork/press-releases/2012/defendant-charged-in-massive-internet-fraud-scheme-extradited-from-estonia-appeared-in-manhattan-federal-court. 2012.

[6] LawFuel Editors. *Massive Internet Fraud Nets Extradicted Estonian Defendant at Least $14 Million*. http://www.lawfuel.com/massive-internet-fraud-nets-extradicted-estonian-defendant-least-14-million. 2014.

[7] Paul Pearce et al. "Characterizing Large-Scale Click Fraud in ZeroAccess". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 141–152. ISBN: 978-1-4503-2957-6.

[8] Daniel Lowd and Christopher Meek. "Good Word Attacks on Statistical Spam Filters". In: *CEAS*. 2005.

[9] Gregory L Wittel and Shyhtsun Felix Wu. "On Attacking Statistical Spam Filters". In: *CEAS*. 2004.

[10] Charles Smutz and Angelos Stavrou. “Malicious PDF Detection Using Metadata and Structural Features”. In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM. 2012, pp. 239–248.

[11] Nedim Rndic and Pavel Laskov. “Practical Evasion of a Learning-Based Classifier: A Case Study”. In: *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE. 2014, pp. 197–211.

[12] Weilin Xu, Yanjun Qi, and David Evans. “Automatically Evading Classifiers”. In: *Proceedings of the 2016 Network and Distributed Systems Symposium*. 2016.

[13] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. “I Am Robot:(deep) Learning to Break Semantic Image Captchas”. In: *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE. 2016, pp. 388–403.

[14] Amir Globerson and Sam Roweis. “Nightmare at Test Time: Robust Learning by Feature Deletion”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 353–360.

[15] Nicolas Papernot et al. “The Limitations of Deep Learning in Adversarial Settings”. In: *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE. 2016, pp. 372–387.

[16] Nicholas Carlini and David Wagner. “Towards evaluating the robustness of neural networks”. In: *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE. 2017.

[17] Prahlad Fogla and Wenke Lee. “Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques”. In: *Proceedings of the 13th ACM conference on Computer and communications security*. ACM. 2006, pp. 59–68.

[18] David Wagner and Paolo Soto. “Mimicry Attacks on Host-Based Intrusion Detection Systems”. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM. 2002, pp. 255–264.

[19] Battista Biggio et al. “Is Data Clustering in Adversarial Settings Secure?” In: *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM. 2013, pp. 87–98.

[20] Battista Biggio et al. “Poisoning Behavioral Malware Clustering”. In: *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*. ACM. 2014, pp. 27–36.

[21] Wei Meng, Ruian Duan, and Wenke Lee. “DNS Changer Remediation Study”. In: *M3AAWG 27th General Meeting*. 2013.

[22] *TDSS/TDL4 Domain Names*. http://www.cc.gatech.edu/~ychen462/files/misc/tdssdomains.pdf.

[23] ClickZ. *Fake Display Ad Impressions Comprise 30% of All Online Traffic [Study]*. http://bit.ly/2e3HdCZ.

[24] Association of National Advertisers. *The Bot Baseline: Fraud in Digital Advertising*. http://bit.ly/1PKe769.

[25] Manos Antonakakis et al. "From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware". In: *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 2012, pp. 491–506.

[26] Yacin Nadji et al. "Connected Colors: Unveiling the Structure of Criminal Networks". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg. 2013, pp. 390–410.

[27] Ulrich Bayer et al. "Scalable, Behavior-Based Malware Clustering." In: *NDSS*. Vol. 9. 2009, pp. 8–11.

[28] Roberto Perdisci, Wenke Lee, and Nick Feamster. "Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces". In: *NSDI*. Vol. 10. 2010, p. 14.

[29] Terry Nelms et al. "Towards Measuring and Mitigating Social Engineering Software Download Attacks". In: *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association. 2016, pp. 773–789.

[30] Luca Invernizzi et al. "Nazca: Detecting Malware Distribution in Large-Scale Networks". In: *NDSS*. Vol. 14. 2014, pp. 23–26.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 701–710.

[32] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, pp. 855–864.

[33] Vincent D Blondel et al. "Fast Unfolding of Communities in Large Networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008.

[34] Erzsébet Ravasz et al. "Hierarchical organization of modularity in metabolic networks". In: *science* 297.5586 (2002), pp. 1551–1555.

[35] Marta Sales-Pardo et al. “Extracting the hierarchical organization of complex systems”. In: *Proceedings of the National Academy of Sciences* 104.39 (2007), pp. 15224–15229.

[36] Gergely Palla et al. “Uncovering the overlapping community structure of complex networks in nature and society”. In: *Nature* 435.7043 (2005), pp. 814–818.

[37] Mark Braverman et al. “ETH hardness for densest-k-subgraph with perfect completeness”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 1326–1341.

[38] Ulrike Von Luxburg. “A Tutorial on Spectral Clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[39] Raymond B Cattell. “The scree test for the number of factors”. In: *Multivariate behavioral research* 1.2 (1966), pp. 245–276.

[40] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. “Singular value decomposition and principal component analysis”. In: *A practical approach to microarray data analysis*. Springer, 2003, pp. 91–109.

[41] Kevin J Lang. “Fixing Two Weaknesses of the Spectral Method”. In: *Advances in Neural Information Processing Systems* (2005).

[42] Kurucz, Miklós and Benczúr, András A. “Geographically Organized Small Communities and the Hardness of Clustering Social Networks”. In: *Data Mining for Social Network Data*. Springer, 2010.

[43] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[44] Paul Barford et al. “Adscape: Harvesting and analyzing online display ads”. In: *Proceedings of the 23rd international conference on World wide web*. ACM. 2014.

[45] Juan Miguel Carrascosa et al. “I Always Feel Like Somebody’s Watching Me. Measuring Online Behavioural Advertising”. In: *ACM CoNEXT*. 2015.

[46] Bhanu C Vattikonda et al. “Empirical Analysis of Search Advertising Strategies”. In: *Internet Measurement Conference (IMC)*. 2015.

[47] Alexander Tuzhilin. *The Lanes Gift v. Google Report (2006)*.

[48] Google. *Ad Traffic Quality Resource Center*. `http://www.google.com/ads/adtrafficquality/`.

[49] Tina Kelleher. *How Microsoft Advertising Helps Protect Advertisers from Invalid Traffic*. http://advertise.bingads.microsoft.com/en-us/blog/26235/how-microsoft-advertising-helps-protect-advertisers-from-invalid-traffic.

[50] Google. *How Google uses conversion data*. https://support.google.com/adwords/answer/93148?hl=en.

[51] Google. *About smart pricing*. https://support.google.com/adwords/answer/2604607?hl=en.

[52] James Wyke. *ZeroAccess*. http://sophosnews.files.wordpress.com/2012/04/zeroaccess2.pdf. 2012.

[53] Interactive Advertising Bureau. *Real-Time Bidding (RTB) Project*. https://www.iab.com/guidelines/real-time-bidding-rtb-project/.

[54] Brett Stone-Gross et al. "Understanding Fraudulent Activities in Online Ad Exchanges". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*.

[55] Tommy Blizard and Nikola Livic. "Click-fraud Monetizing Malware: A Survey and Case Study". In: *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. IEEE. 2012, pp. 67–72.

[56] Alan Neville. *Waledac Reloaded: Trojan.Rloader.B*. http://www.symantec.com/connect/blogs/waledac-reloaded-trojanrloaderb. 2013.

[57] Qing Zhang et al. "Got Traffic? An Evaluation of Click Traffic Providers". In: *Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality*. ACM. 2011, pp. 19–26.

[58] Aleksandr Matrosov. *TDSS part 1 through 4*. http://resources.infosecinstitute.com/tdss4-part-1/. 2011.

[59] P. Mockapetris. *Domain Names - Concepts and Facilities*. http://www.ietf.org/rfc/ffc1034.txt. 1987.

[60] P. Mockapetris. *Domain Names - Implementation and Specification*. http://www.ietf.org/rfc/rfc1035.txt. 1987.

[61] Guy Bruneau. *DNS Sinkhole*. http://www.sans.org/reading_room/whitepapers/dns/dns-sinkhole_33523. 2010.

[62] Manos Antonakakis et al. *Unveiling the Network Criminal Infrastructure of TDSS/TDL4 DGAv14: A case study on a new TDSS/TDL4 variant.* Technical Report, Damballa Inc., Georgia Institute of Technology (GTISC). 2012.

[63] Messaging Anti-Abuse Working Group and others. "M3AAWG Best Practices for the use of a Walled Garden". In: *San Francisco, CA* (2007).

[64] Dan Pelleg and Andrew W. Moore. "X-means: Extending K-means with Efficient Estimation of the Number of Clusters". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 727–734. ISBN: 1-55860-707-2.

[65] Kevin Springborn and Paul Barford. "Impression Fraud in Online Advertising via Pay-Per-View Networks". In: *Proceedings as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. 2013, pp. 211–226.

[66] Mila Parkour. *Collection of Pcap files from malware analysis*. `http://contagiodump.blogspot.com/2013/04/collection-of-pcap-files-from-malware.html`. 2013.

[67] Eugene Rodionov and Aleksandr Matrosov. "The Evolution of TDL: Conquering x64". In: *ESET, June 2011* ().

[68] Vacha Dave, Saikat Guha, and Yin Zhang. "Measuring and Fingerprinting Click-Spam in Ad Networks". In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM. 2012, pp. 175–186.

[69] Brett Stone-Gross et al. "Your Botnet is my Botnet: Analysis of a Botnet Takeover". In: *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, pp. 635–647.

[70] Rob J Hyndman. *Transforming data with zeros*. `http://robjhyndman.com/hyndsight/transformations/`. 2010.

[71] David Meyer et al. *University of Oregon route views uroject*. 2005.

[72] Howard Beales. "The value of behavioral targeting". In: *Network Advertising Initiative* (2010).

[73] United States District Court. *Sealed Indictment*. `http://www.wired.com/images_blogs/threatlevel/2011/11/Tsastsin-et-al.-Indictment.pdf`. 2011.

[74] A Matrosov and E Rodionov. “TDL3: The Rootkit of All Evil”. In: *ESET, Security 2009* ().

[75] Sergey Golovanov Igor Soumenkov. *TDL4 - Top Bot*. http://securelist.com/analysis/36152/tdl4-top-bot/. 2011.

[76] Vyacheslav Rusakov. *TDSS. TDL-4*. http://securelist.com/analysis/36339/tdss-tdl-4/. 2011.

[77] Guy Bruneau and Rick Wanner. *DNS Sinkhole*. Tech. rep. SANS Institute InfoSec Reading Room, 2010.

[78] Christian Rossow et al. “SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets”. In: *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE. 2013, pp. 97–111.

[79] Zhou Li et al. “Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising”. In: *Proceedings of the 2012 ACM conference on Computer and Communications Security*. ACM. 2012, pp. 674–686.

[80] Neil Daswani and Michael Stoppelman. “The anatomy of Clickbot.A”. In: *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. USENIX Association. 2007.

[81] openrtb.info. *OpenRTB: Documentation and Issue tracking for the OpenRTB Project*. http://openrtb.github.io/OpenRTB/. 2014.

[82] Zeus Tracker. *Zeus IP & Domain Name Block List*. https://zeustracker.abuse.ch. 2009.

[83] *I.T. Mate List*. http://vurldissect.co.uk/daily.asp.

[84] *Malc0de Database*. http://malc0de.com/bl/BOOT.

[85] *Malware Domain List*. https://www.malwaredomainlist.com/.

[86] *sagadc.org list*. http://dns-bh.sagadc.org/domains.txt.

[87] *Hphosts List*. http://hosts-file.net/?s=Download.

[88] *SANS ISC Feeds*. https://isc.sans.edu/feeds/.

[89] Department of Homeland Security. *Trusted Cyber Risk Research Data Sharing*. https://www.dhs.gov/csd-impact.

[90] *PassiveTotal: RiskIQ*. https://www.passivetotal.org/.

[91] Malware Tips. *How to remove Websearch.searc-hall.info*. http://bit.ly/2e9qyKw.

[92] Malware Tips. *Remove Sl.now-update-check.com virus*. http://bit.ly/2dm1LWp.

[93] Alexa. *The Web Information Company*. http://www.alexa.com/. 2007.

[94] *EasyList*. https://easylist-downloads.adblockplus.org/easylist.txt.

[95] *Mozilla Public Suffix List*. https://publicsuffix.org/list/. 2015.

[96] VirusTotal. *Antivirus scan*. https://goo.gl/jU0b0b. 2014.

[97] TrendMicro, Inc. *Threat Encyclopedia: TROJ_LEMIR.CS*. https://goo.gl/8ryRjK. 2012.

[98] VirusTotal. *Antivirus scan*. https://goo.gl/s97XI5. 2015.

[99] VirusTotal. *IP address information*. https://goo.gl/ifLvT5. 2015.

[100] Brad Miller et al. "What's clicking what? techniques and innovations of today's clickbots". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2011, pp. 164–183.

[101] Sumayah A Alrwais et al. "Dissecting ghost clicks: ad fraud via misdirected human clicks". In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM. 2012, pp. 21–30.

[102] Vacha Dave, Saikat Guha, and Yin Zhang. "ViceROI: catching click-spam in search ad networks". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, pp. 765–776.

[103] Tian Tian et al. "Crowd fraud detection in internet advertising". In: *Proceedings of the 24th International Conference on World Wide Web*. ACM. 2015, pp. 1100–1110.

[104] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. "Detectives: detecting coalition hit inflation attacks in advertising networks streams". In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, pp. 241–250.

[105] Kurt Thomas et al. “Ad injection at scale: Assessing deceptive advertisement modifications”. In: *2015 IEEE Symposium on Security and Privacy*.

[106] Alexandros Kapravelos et al. “Hulk: Eliciting malicious behavior in browser extensions”. In: *23rd USENIX Security Symposium (USENIX Security)*. 2014.

[107] Xinyu Xing et al. “Understanding Malvertising Through Ad-Injecting Browser Extensions”. In: *Proceedings of the 24th International Conference on World Wide Web*. 2015.

[108] Ling Huang et al. “Adversarial Machine Learning”. In: *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM. 2011, pp. 43–58.

[109] Kymie MC Tan, Kevin S Killourhy, and Roy A Maxion. “Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits”. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2002, pp. 54–73.

[110] Florian Tramèr et al. “Stealing Machine Learning Models via Prediction Apis”. In: *USENIX Security*. 2016.

[111] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. “Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples”. In: *arXiv preprint arXiv:1605.07277* (2016).

[112] Daniel Lowd and Christopher Meek. “Adversarial Learning”. In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM. 2005, pp. 641–647.

[113] CYBERWARZONE. *30 Malicious IP List and Block Lists Providers 2015*. `http://cyberwarzone.com/30-malicious-ip-list-and-block-lists-providers-2015/`. Accessed in May 2017.

[114] *OpenIOC DB*. `https://openiocdb.com/`. Accessed in May 2017.

[115] *IOC Bucket*. `https://www.iocbucket.com/`. Accessed in May 2017.

[116] *Microsoft Azure*. `https://azure.microsoft.com`. Accessed in May 2017.

[117] *Amazon Machine Learning*. `https://aws.amazon.com/machine-learning`. Accessed in May 2017.

[118] *BigML*. `https://bigml.com/`. Accessed in May 2017.

[119] *PredictionIO*. `http://prediction.io`. Accessed in May 2017.

[120] *Google Cloud Prediction API*. https://cloud.google.com/prediction/docs/. Accessed in May 2017.

[121] Yizheng Chen et al. "Financial Lower Bounds of Online Advertising Abuse". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 231–254.

[122] Manos Antonakakis et al. "Building a Dynamic Reputation System for DNS". In: *the Proceedings of 19th USENIX Security Symposium (USENIX Security '10)*. 2010.

[123] Konrad Rieck et al. "Automatic analysis of malware behavior using machine learning". In: *Journal of Computer Security* 19.4 (2011), pp. 639–668.

[124] Jimeng Sun et al. "Neighborhood formation and anomaly detection in bipartite graphs". In: *Data Mining, Fifth IEEE International Conference on*. IEEE. 2005, 8–pp.

[125] Kun Liu and Evimaria Terzi. "Towards identity anonymization on graphs". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pp. 93–106.

[126] Angsheng Li and Pan Peng. "The small-community phenomenon in networks". In: *Mathematical Structures in Computer Science* 22.03 (2012), pp. 373–407.

[127] Johannes Bader. *Domain Generation Algorithms*. https://github.com/baderj/domain_generation_algorithms. 2017.

[128] Network X. *Community API*. http://perso.crans.org/aynaud/communities/api.html. Accessed in May 2017.

[129] Hermit Dave. *Frequency Words in Subtitles*. https://github.com/hermitdave/FrequencyWords/tree/master/content/2016/en. Accessed in May 2017.

[130] Duen Horng Polo Chau et al. "Polonium: Tera-scale graph mining and inference for malware detection". In: *Proceedings Of The 2011 Siam International Conference On Data Mining*. SIAM. 2011, pp. 131–142.

[131] Philipp Trinius et al. "A malware instruction set for behavior-based analysis". In: (2009).

[132] Marcos Sebastián et al. "Avclass: A Tool for Massive Malware Labeling". In: *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer. 2016, pp. 230–253.

[133] Yizheng Chen et al. “DNS Noise: Measuring the Pervasiveness of Disposable Domains in Modern DNS Traffic”. In: *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE. 2014, pp. 598–609.

[134] Yacin Nadji et al. “Beheading Hydras: Performing Effective Botnet Takedowns”. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, pp. 121–132.